2600 シリーズ システム ソースメータ

リファレンス マニュアル

260x-901-01 Rev. A / May 2005

KEITHLEY

A GREATER MEASURE OF CONFIDENCE

保証規定

Keithley Instruments, Inc. は、本製品が出荷後 1 年間の期間にわたり材質もしくは製造技術上の瑕疵によって欠陥を生じないことを保証します。

Keithley Instruments, Inc. は以下の項目が出荷後 90 日の期間にわたり欠陥を生じないことを保証します：プローブ、ケーブル、充電式電池、ディスケット、及びドキュメント。

万一保証期間内に製品に問題が発生しました場合には、欠陥と判定された製品の修理もしくは交換を行います（どちらを実施するかは弊社の裁量とさせて頂きます)。

本保証規定の実施を請求されます場合は、最寄の弊社代理店、もしくは本社（Keithley 本社、Cleveland、Ohio）へ書面もしくは電話でご連絡ください。遅滞なく事態の処理、および製品の返送に関する連絡を差し上げます。該当製品は送料前払いにて指定されたサービス施設へお送りください。修理の完了した製品は輸送量当社負担にてお客様へ返却されます。修理 / 交換された製品に適用される保証期間は当初の保証の有効残余期間、もしくは少なくとも 90 日です。

保証の制限

Keithley からの書面による許諾なしに実施された改造、あるいは製品および付随する部品の誤使用に起因する欠陥は本保証規定の対象外となります。また、ヒューズ、ソフトウェア、使い捨て電池、電池の液漏れに起因する損傷、および通常予期される磨耗、取扱説明書の指示に従われなかったことに起因する故障も保証の対象となりません。

本保証規定は、本製品を特定の目的に使用した場合の適性や商品価値を示唆したものを含め、書面によると含意によるとを問わず、他のあらゆる保証に優先します。損害に対して本保証が提供する救済は購入者が得られる唯一かつ排他的な救済策です。

KEITHLEY INSTRUMENTS, INC. はもとよりその従業員も、装置もしくはソフトウェアの使用に起因する直接的、間接的、特異的、あるいは偶発的、従属的に発生し得る損害に対して、例え KEITHLEY INSTRUMENTS, INC. がそのような可能性について事前の連絡を受けていたとしても、その責を負いません。保証の対象外となる損害には撤去および設置に係わる費用、要員の怪我に付随して継続的に発生する損失、または財産の毀損等が含まれ、かつこれだけに限定されるものではありません。

KEITHLEY

A GREATER MEASURE OF CONFIDENCE

Keithley Instruments, Inc.

Corporate Headquarters • 28775 Aurora Road • Cleveland, Ohio 44139
440-248-0400 • Fax: 440-248-6168 • 1-888-KEITHLEY (534-8453) • www.keithley.com

12/04

2600 シリーズ システム ソースメータ
リファレンス マニュアル

©2005, Keithley Instruments, Inc.
不許複製
Cleveland, Ohio, U.S.A.
第 1 刷：2005 年 5 月
資料番号：260x-901-01 Rev. A

マニュアル印刷履歴

本マニュアルに追加された改訂と補遺の印刷履歴を以下に示します。マニュアルが更新されるたびに版数を示すアルファベットが A,B,C…の順に変化してゆきます。正式な改訂と次の改訂の間に発生した重要な変更で、かつお客様に遅滞なくお知らせする必要のある情報は随時発行される補遺に記載されています。補遺には連番が付けられます。新しい改訂版を発行するときは、その前の版数のマニュアルに付随する補遺の内容がすべて新しい版に組み込まれます。この印刷履歴ページも各改訂版ごとに内容を更新して添付されます。

改訂版 A（資料番号 260x-901-01）.....2005 年 5 月

すべてのケースレー製品名は Keithley Instruments, Inc. の商標または登録商標です。
それ以外のブランド名はそれぞれの権者に帰属する商標 / 登録商標です。

KEITHLEY

安全上の注意事項

本製品および付随する装置をご使用になられる前に、以下に説明する安全上の注意事項を必ずご確認ください。装置やアクセサリの中には通常の使用条件では危険な高電圧を使用しないものもありますが、そのような装置であっても置かれる状況によっては危険な状態が発生する可能性があります。

本製品は感電の危険を良く認識し、事故防止に必要な安全上の注意事項を熟知した人による使用を前提としています。製品を使用される前に設置・操作・保守に関する説明をよく読み、それに従ってください。製品仕様の詳細についてはマニュアルをご覧ください。

指定された以外の方法で製品を使用された場合は、装置が本来備える保護機能を損う可能性があります。

本装置を使用されるのは次のような方々です。

責任者（責任団体）は装置の使用と保守に責任を負う個人またはグループであり、装置が本来の仕様と動作限界の範囲内で適正に運用され、オペレータが適切な教育を受けることに対して責任を負います。

オペレータは目的の機能を実現するために製品を使用します。オペレータは電気的な安全保持および装置の適正な運用について教育を受けていなければならず、電気ショックや通電回路に直接触れぬように保護されている必要があります。

保守担当者は製品を正しく動作させるために必要な所定の作業を行います（例えば、電源電圧の設定、消耗部品の交換など）。具体的な保守作業の内容についてはマニュアルをご覧ください。保守担当者が実施できる項目であるかどうかはそれぞれの作業説明に明記してあります。該当しない項目についてはサービス担当者にお任せください。

サービス担当者は安全に装置を設置し、製品の修理を行います。このため、サービス担当者は活線作業実施のための教育を受けていなければなりません。設置やサービス作業を行えるのは適正な教育を受けたサービス担当者だけです。

ケースレー社の製品は国際電気標準会議（IEC）規格 IEC 60664 に従って測定カテゴリーI、または測定カテゴリー II に位置付けられた電気信号を対象として設計されています。

測定、コントロール、およびデータ I/O 信号の殆どは測定カテゴリー I に分類されるものであり、電源電圧や大きな過渡電圧を伴う電源に直接接続することは許されません。測定カテゴリー II の接続では、現場の AC 電源接続に往々に見られる大きな過渡電圧の印加に対する保護が必要となります。マニュアルに特段の注記または説明がない場合は、すべての測定・コントロール・I/O 接続はカテゴリー I 信号源に接続するものと見なしてください。

感電の危険性が存在する場合は特別な注意が必要です。ケーブル接続ジャックや試験ジグには人命にかかわる高電圧が印加されていることがあります。米国規格協会（ANSI）によれば、電圧レベルが 30V RMS、42.4V（ピーク値）、または 60VDC を超える場合は常に感電の危険が存在すると規定しています。未知回路を測定しようとするときは、常に危険な高電圧が存在するものとして作業を行ってください。

オペレータは作業中常に感電から保護されていなければなりません。責任者（団体）はオペレータが危険箇所に触れないよう、絶縁されているようにすべての接続ポイントを処置する必要があります。ただし、場合によっては人が操作できるように接続個所をあえて露出させなければならないことがあります。オペレータは、このような状況でも感電事故から自分を保護できるように教育されていなければなりません。しかし、1000V を超える電圧で動作する可能性のある回路については、絶対に回路の導電部位を露出させないでください。

スイッチングカードを電力制限機能のない電源回路に直接接続しないでください。これらのカードは本来インピーダンスで制限された電源への接続を想定したものです。スイッチングカードは絶対に AC 電源に直結しないでください。スイッチングカードに電源を接続するときは、過大な電流や電圧がカードに印加されないようにする保護デバイスを装着してください。

装置を動作させる前に、電源コードが正しく接地されたコンセントに接続されていることを確認してください。接続ケーブルや試験導線、ジャンパ等に磨耗や割れ目、断線などがないか検査してください。

電源コードへのアクセスが難しい場所（ラックなど）に装置を設置する場合は、主電源を遮断できる独立したデバイスを装置のできるだけ近くで、かつオペレータが容易に操作できる位置に設けてください。

確実に安全を確保するため、被試験回路が通電されている間は製品、試験ケーブル、その他の装置には手を触れないようにしてください。次のような作業を行う場合は必ず事前に全回路の電源を切り、コンデンサを放電させてください。ケーブル / ジャンパの接続や取り外し、スイッチングカードの装着 / 取り外し、ジャンパの取付け / 取り外しなどの内部設定変更。

被試験回路のコモン側や電源 GND へ電流を流す経路となりえる物体には触らないでください。測定を行うときは手が濡れていないことを確認し、測定対象の電圧に十分耐える乾燥した非導電性の床の上に立って作業してください。

装置およびアクセサリはその仕様と指定された操作法に従って使用してください。これが守られない場合は装置を安全に使用していただくことはできません。

装置およびアクセサリの最大信号レベルを絶対に超えないようにしてください。この値は装置仕様と操作法に定義されており、さらに装置や試験ジグのパネル、スイッチングカードにも表記されています。

製品が装着しているヒューズを交換するときは、火災事故に対して引き続き同等レベルの保護を与えるために必ず型式と定格が同じヒューズを使用してください。

筐体端子は測定回路をシールドするための接続に使用し、安全接地 GND としての接続には使用しないでください。

試験ジグを使用する場合、被試験デバイスの通電中は必ず蓋を閉じてください。安全に作業するためにはインターロック付きの蓋を使用してください。

装置に ⏚ シンボルが表記された端子が付いている場合は、その端子をユーザ用資料で推奨されたケーブルを用いて保安接地してください。

装置上に ⚠ シンボルが表記された個所については、マニュアルに記載の操作説明を必ずご覧ください。

装置上に ⚡ シンボルが表記された個所は、1000 V またはそれ以上（通常およびコモンモード電圧の両方を含む）の電圧が発生、または測定できる個所であることを示します。使用者がこのような電圧に直接触れないようにするため、標準的な対策を施して安全を確保してください。

⏚ シンボルは装置筐体への接続端子であることを示します。

マニュアルの中で「警告」という見出しに続く文は、使用する人に傷害（場合によっては致命的な）を引き起こす可能性のある危険性について説明しています。説明の内容を熟読理解してから、該当する操作を実施してください。

マニュアルの中で「注意」という見出しに続く文は、装置の損傷に結びつく危険事項を説明しています。このような損傷は保証規定を無効にすることがありますからご注意ください。

装置およびアクセサリを人体に接続しないでください。

保守作業を行うときは必ず事前に電源ケーブルとすべての試験ケーブルを取り外してください。

感電や火災事故などへの保護機能を維持するため、主要回路要素（電源トランス、試験導線、入力ジャックなど）を交換する必要が生じた場合は、必ずケースレーから純正品を購入してください。ヒューズについては、該当安全規格を満たし、かつ型式と定格が一致するものであれば標準品を使用できます。それ以外の部品で装置の安全に直接関係しないものについては、本来の部品と同等な他社製品を購入していただいてかまいません。（ただし一部の指定された部品については、製品の確度と機能を保つために、ケースレーから購入していただく必要があります。）交換部品の適用性についてご不明の点があるときは、ケースレーの支社、代理店にお問い合わせください。

装置をクリーニングするときは湿らせた布、または水性の洗浄剤を使用してください。クリーニングできるのは装置の外周りだけです。装置自体（内部）は直接クリーナーで触らないでください、また、装置表面や内部に液体が入らないように注意してください。ケースや筐体のない裸の回路基板（コンピュータに装着するデータ収集基板など）は、指示に従って適切に操作されている限りにおいてクリーニングの必要はありません。基板が汚れ、それによって機能に影響が生じている場合は、基板を弊社に返送してクリーニング/サービスを依頼してください。

目次

1 ご使用前に

はじめに 1-2
機能と特徴 1-2
本マニュアルのセクション構成 1-2
一般事項 1-3
保証について 1-3
連絡先について 1-3
安全に関するシンボルと用語 1-3
開梱と検査 1-4
オプションとアクセサリ 1-5
ユーザマニュアル 1-6
リファレンス マニュアル 1-6
前面パネルと背面パネルに慣れましょう 1-6
前面パネルのまとめ 1-6
背面パネルのまとめ 1-6
冷却用通気口 1-12
電源を入れる 1-13
電源ラインの接続 1-13
起動シーケンス 1-14
ブザー 1-14
表示モード 1-15
編集用コントロール要素 1-16
ソースとコンプライアンスの編集 1-16
メニュー ナビゲーション 1-17
メニュー 1-18
メインメニュー 1-18
設定メニュー 1-18
インターフェイスの選択 1-21
GPIB インターフェイス 1-21
RS-232 インターフェイス 1-21
エラー / ステータスメッセージ 1-21
デフォルト設定 1-22
前面パネルの設定 1-22
リモート操作用のセットアップ 1-23
リモートプログラミング 1-25
読み込み要求 1-25
コマンド設定要求 1-25

2 TSP プログラミングの基本

はじめに 2-2
TSP (Test Script Processor) 2-2
ランタイム環境 2-3
クエリ 2-3
スクリプト 2-3
名前付きスクリプト 2-4
関数 2-4
関数を生成するスクリプト 2-5
プログラミングの概要 2-6
チャンクとは何か？ 2-6
スクリプトとは何か？ 2-6
ランタイム環境 2-7
不揮発性メモリ 2-8
TSP のプログラミングレベル 2-8
スクリプトのプログラムモデル 2-8
TSP ソフトウェアのインストール 2-10
システム接続 2-10
GPIB 2-10
RS-232 2-11
Test Script Builder の使用 2-12
Test Script Builder の起動 2-14
通信回線を開く 2-15
スクリプトの作成と変更 2-17
スクリプト起動条件の設定 2-22
スクリプトの起動 2-25
TSP ファイルの実行 2-26
260x 型装置からスクリプトを読み出す 2-26
インスツルメンツコンソール (Instrument Console) 2-28
ファイル管理タスク 2-34
コマンドと文の送信 2-38
電圧と電流の発生と測定 2-38
デジタル I/O ポートの読み出しと書き込み 2-38
ユーザ定義メッセージの表示 2-38
ユーザスクリプト 2-39
スクリプト例 2-39
ユーザスクリプトの作成 2-42
ユーザスクリプトの保存 2-43
ユーザスクリプトの実行 2-43
ユーザスクリプトの変更 2-46
スクリプトの管理 2-46
ファクトリスクリプト 2-48
ファクトリスクリプトの実行 2-48
ファクトリスクリプトの変更 2-48

相違点：リモート vs. ローカル状態 2-49
メモリに関する注意事項 2-51
TSL (Test Script Language) リファレンス 2-52
はじめに 2-52
予約語 2-52
変数と変数型 2-52
演算子 2-53
テーブル / 配列 2-53
関数 2-54
優先度 2-55
論理演算子 2-55
連接 2-56
分岐 2-57
ループ制御 2-58
標準ライブラリ 2-59

3 DUT 試験の接続
入出力端子ブロック 3-2
入出力 LO との筐体の接地 3-3
センシング方法 3-5
2 線式ローカルセンシング 3-5
4 線式リモートセンシング 3-5
センスモード選択 3-6
SMU の複数接続 3-7
ガードとシールド 3-8
ガード 3-8
ノイズシールド 3-9
安全シールド 3-10
シールドとガードの併用 3-11
テストフィクスチャ 3-12
SMU をフローティングにする 3-13
出力 OFF 状態 3-15
出力 OFF 状態の選択 3-16

4 基本操作
概要 4-2
操作のまとめ 4-2
ソースメジャー機能 4-2
コンプライアンスリミット 4-3
コンプライアンスリミットの設定 4-4
基本回路構成 4-5

操作上の注意事項 4-7
ウォームアップ 4-7
オートゼロ 4-7
NPLC キャッシング 4-7
トリガ 4-9
トリガのタイプ 4-9
測定トリガ 4-10
前面パネルトリガ 4-11
リモートトリガ 4-12
基本的なソースメジャー手順 4-13
前面パネルからソースメジャー手順を実行 4-13
ソースメジャー手順のリモート操作 4-14
測定（メジャー）専用 4-16
シンク操作 4-17
抵抗測定 4-18
抵抗値の計算 4-18
抵抗測定のレンジ設定 4-18
基本的な抵抗測定操作 4-18
抵抗センシング 4-19
センシング方法の選択 4-20
抵抗測定のリモートプログラミング 4-21
電力測定 4-22
電力の演算 4-22
電力測定の基本手順 4-22
電力測定のリモートプログラミング 4-23

5 スイープ操作

概要 5-2
このセクションのまとめ 5-2
スイープのまとめ 5-2
スイープ特性 5-3
線形階段スイープ 5-3
対数階段スイープ 5-4
パルススイープ 5-6
カスタム（リスト）スイープ 5-7
スイープ測定値の保存 5-8
スイープ関数 5-8
階段スイープ関数 5-9
パルススイープ関数 5-10
カスタムスイープ関数 5-10
スイープ実行 5-11
前面パネル 5-11
スイーププログラミングの例 5-11
カスタムスイープの例 5-14

6 レンジ、桁数、速度、リラティブ、およびフィルタ

概要 6-2
レンジ 6-2
選択可能なレンジ 6-2
ソースの最大値と読み取り値 6-3
レンジ設定に関する制約事項 6-3
手動レンジ切換え 6-3
自動レンジ切換え 6-3
低域側のレンジリミット 6-4
レンジ設定についての注意事項 6-4
レンジ設定のプログラミング 6-5
桁数 6-7
表示分解能の設定 6-7
表示桁数のリモートプログラミング 6-7
速度 6-8
速度設定 6-9
速度設定のリモートプログラミング 6-9
リラティブ 6-10
前面パネルからの リラティブ 操作 6-10
リラティブ のリモートプログラミング 6-11
フィルタ 6-12
フィルタタイプ 6-12
応答時間についての注意事項 6-12
前面パネルからのフィルタコントロール 6-12
フィルタのリモートプログラミング 6-15

7 バッファ（データ保存）

概要 7-2
データ保存のまとめ 7-2
前面パネル操作によるデータ保存 7-2
バッファの設定 7-2
バッファ設定メニュー 7-3
読み取り値の保存 7-3
読み取り値の呼び出し 7-4
リモート操作によるデータ保存 7-5
データ保存コマンド 7-5
バッファの読み出し 7-6
時間と日付の値 7-10
バッファステータス 7-10
バッファの動的割り付け 7-11
バッファのプログラミング例 7-12

8 ソースメジャーの概念

概要 8-2
コンプライアンスリミット 8-2
最大コンプライアンス 8-2
コンプライアンスの原理 8-3
スイープ波形 8-4
階段スイープ 8-4
パルススイープ 8-5
過熱保護 8-6
過熱予防に役立つパワー計算式 8-6
パワー計算 8-6
動作の限界 8-7
ソースかシンクか 8-7
電流ソース (I-Source) の動作限界 8-8
電圧ソース（V -Source）の動作限界 8-12
電流ソース - 電流測定、電圧ソース - 電圧測定 8-16
基本回路構成 8-16
電流ソース 8-16
電圧ソース 8-17
測定のみ（V または I） 8-18
ガード 8-20
ガードの概要 8-20
ガードの接続 8-20
パルスの概念 8-23
パルス周期 8-23
パルスのライズ / フォールタイム 8-23
パルスのデューティサイクル 8-24

9 システムの拡張 (TSP-Link)

概要 9-2
マスタとスレーブ 9-2
システム構成 9-2
接続 9-3
初期化 9-3
ノード番号の割り付け 9-3
TSP-Link のリセット 9-4
拡張されたシステムの使用 9-6
ノードへのアクセス 9-6
システムの挙動 9-7

10 デジタル I/O とインターロック

概要 10-2
デジタル I/O ポート 10-2
ポート構成 10-2
デジタル I/O の回路構成 10-4
デジタル I/O ラインのコントロール 10-4
インターロック 10-8
概要 10-8
操作 10-8
コントロール 10-9

11 通信インターフェイス

概要 11-2
インターフェイスの選択 11-2
GPIB の操作 11-3
GPIB 規格 11-3
GPIB 接続 11-3
一次アドレス 11-5
終端記号 11-6
一般バスコマンド 11-7
REN (remote enable) 11-7
IFC (interface clear) 11-7
LLO (local lockout) 11-8
GTL (go to local) 11-8
DCL (device clear) 11-8
SDC (selective device clear) 11-8
GET (group execute trigger) 11-8
SPE, SPD (serial polling) 11-8
前面パネルからの GPIB 操作 11-9
エラー / ステータスメッセージ 11-9
GPIB ステータス表示ランプ 11-9
LOCAL キー 11-10
RS-232 インターフェイスの操作 11-10
RS-232 インターフェイス パラメータの設定 11-10
データの送受信 11-11
終端記号 11-11
ボーレート 11-12
データビットとパリティ 11-12
フローコントロール（信号ハンドシェーク） 11-12
RS-232 接続 11-13
エラーメッセージ 11-14

12 装置コントロールライブラリ

コマンドのプログラミングに関する注意事項 12-2
規約 12-2
関数と属性値 12-3
TSP-Link のノード 12-5
論理装置 12-5
読み込みバッファ 12-6
時間と日付の値 12-8
ICL 関数と属性値一覧 12-9
beeper 関数と属性値 12-11
bit 関数 12-12
delay 関数 12-18
digio 関数と属性値 12-19
display 関数と属性値 12-24
errorqueue 関数と属性値 12-40
exit 関数 12-42
format 属性値 12-42
gpib 属性値 12-45
localnode 属性値 12-46
makegetter 関数 12-48
opc 関数 12-50
printbuffer 関数と printnumber 関数 12-51
reset 関数 12-53
serial 関数と属性値 12-53
setup 関数と属性値 12-56
smuX 関数と属性値 12-57
timer 関数 12-86
trigger 関数 12-87
tsplink 関数と属性値 12-88
userstring 関数 12-89
waitcomplete 関数 12-91

13 ファクトリスクリプト

はじめに 13-2
ファクトリスクリプト 13-2
KIGeneral 13-2
フラッシュファームウェアのアップグレード 13-13

14 ディスプレイの操作

display 関数と属性値 14-2
ディスプレイの持つ機能 14-3
　ディスプレイ画面 14-3
　測定関数 14-3
　表示分解能 14-4
メッセージ表示 14-4
　画面クリア 14-5
　カーソル位置 14-5
　テキストメッセージの表示 14-6
入力プロンプト 14-8
　メニュー 14-8
　パラメータ入力プロンプト 14-10
表示ランプ 14-11
LOCAL ロックアウト 14-12
テストメニューのローディング 14-13
　ユーザスクリプトの保存 14-13
　USER TESTS メニューの項目追加 14-13
　USER TESTS メニューの項目削除 14-14
　前面パネルからテストを実行 14-14
ディスプレイトリガ 14-15
キープレスコード 14-16
　キーコードを送る 14-16
　キープレスコードの取り込み 14-17

15 性能確認

はじめに 15-2
確認試験の要件 15-2
　環境条件 15-2
　ウォームアップ時間 15-3
　電源ライン 15-3
　推奨試験装置 15-3
　性能確認の限界値 15-4
ファクトリデフォルトの呼び戻し 15-5
確認試験手順の実行 15-5
　試験のまとめ 15-5
　試験実施のための考慮事項 15-5
　ソースレンジと出力値の設定 15-6
　測定レンジの設定 15-6
出力電圧の確度 15-7
電圧測定の確度 15-8
出力電流の確度 15-9
電流測定の確度 15-11

16 校正

はじめに 16-2
環境条件 16-2
　温度と相対湿度 16-2
　ウォームアップ時間 16-2
　供給電源 16-2
校正作業の注意事項 16-3
　校正サイクル 16-3
　校正に使用できる推奨装置 16-4
　校正エラー 16-5
校正 16-5
　校正ステップ 16-5
　校正コマンド 16-6
　校正手順 16-8

17 日常のメンテナンス

はじめに 17-2
電源ヒューズの交換 17-2
前面パネルのテスト 17-3
　キー動作テスト 17-3
　表示パターンテスト 17-4

A 仕様

B エラーメッセージとステータスメッセージ

はじめに B-2
エラーのまとめ B-2
エラーの読み取り B-2

C 共通コマンド

はじめに C-2
共通コマンド C-2
　コマンド要約 C-2
　対応するスクリプトコマンド C-3
　コマンドリファレンス C-4

D ステータスモデル

概要 D-2
ステータスバイトと SRQ D-2
ステータスレジスタセット D-2
キュー D-2
ステータス関数のまとめ D-8
レジスタとキューのクリア D-9
レジスタのプログラミングと読み出し D-10
イネーブルレジスタと遷移レジスタのプログラミング D-10
レジスタの読み出し D-11
ステータスバイトとサービス要求（SRQ） D-11
ステータスバイトレジスタ D-11
サービス要求イネーブルレジスタ D-13
シリアルポーリングと SRQ D-14
SPE、SPD (シリアルポーリング) D-14
ステータスバイトとサービス要求コマンド D-14
イネーブルレジスタと遷移レジスタ D-15
ノードと SRQ イネーブルレジスタのコントロール D-15
ステータスレジスタセット D-17
システムサマリイベントレジスタ（System Summary Event Register） D-17
標準イベントレジスタ（Standard Event Register） D-19
動作イベントレジスタ（Operation Event Register D-21
クエスチョナブルイベントレジスタ（Questionable Event Register） D-24
測定イベントレジスタ（Measurement Event Register） D-26
レジスタのプログラミング例 D-29
キュー D-29
出力キュー D-29
エラーキュー D-29

E 速度仕様試験実施の条件

はじめに E-2
使用する試験システム E-2
概要 E-2
スイープ動作速度 E-2
シングル測定速度 E-3
機能とレンジ切換え速度 E-3
コマンド処理 E-4
スイープ動作速度 E-4
測定値をメモリに書き込み E-5
測定値を GPIB へ転送 E-5
ソース測定値をメモリに書き込み E-6
ソース測定値を GPIB へ転送 E-6

ソース測定合否判定をメモリに書き込み E-6
ソース測定合否判定を GPIB へ転送 E-6
シングル測定速度 E-6
測定値を GPIB へ転送 E-7
ソース測定値を GPIB へ転送 E-7
ソース測定合否判定を GPIB へ転送 E-7
機能 / レンジ切換え速度 E-8
ソースレンジ切換え速度 E-8
測定レンジ切換え速度..... E-8
機能切換え速度 E-8
コマンド処理 E-9

図リスト

1 ご使用前に

図 1-1 2601、2602 型の前面パネル 1-7
図 1-2 2601、2602 型の背面パネル 1-10
図 1-3 表示モード 1-15

2 TSP プログラミングの基本

図 2-1 スクリプトの例 2-7
図 2-2 スクリプトのプログラミングモデル 2-9
図 2-3 GPIB ケーブル 2-10
図 2-4 RS-232 ケーブル（ストレート） 2-11
図 2-5 Test Script Builder 画面例 2-13
図 2-6 通信のオープン / クローズ 2-16
図 2-7 Test Script Builder を使用するスクリプトの作成 / 変更 2-17
図 2-8 プロジェクトフォルダの作成 2-18
図 2-9 Test Script Builder にスクリプトを保存する 2-19
図 2-10 スクリプトファイルの新規作成 2-20
図 2-11 プロジェクトフォルダ / スクリプトファイルの名前の付け替え 2-21
図 2-12 起動条件設定の変更 2-22
図 2-13 Run ダイアログボックスを開く（起動条件設定） 2-23
図 2-14 Run ダイアログボックス（Script Attributes タブ） 2-25
図 2-15 Test Script Builder ツールバーを使用してスクリプトを再起動 2-25
図 2-16 Test Script Builder ツールバーを使用してスクリプトを再起動 2-26
図 2-17 260x 型からスクリプト（ここでは KIGeneral_Script）をインポートする方法 2-27
図 2-18 Instrument Console アイコン 2-28
図 2-19 プログラミング操作に関するタブ : Problems、Tasks、Command Help 2-32
図 2-20 プログラミング操作に関するタブ : Language Help、Bookmarks、Browser View 2-33
図 2-21 ワークスペースランチャとワークスペース ディレクトリ選択ダイアログ 2-35
図 2-22 別なワークスペースフォルダからプロジェクトをインポート 2-36
図 2-23 プロジェクトの削除 2-37

3 DUT 試験の接続

図 3-1 入出力端子ブロック 3-3
図 3-2 入出力 LO と筐体接地端子 3-4
図 3-3 低ノイズ筐体 GND 用バナナジャックと筐体ネジ 3-4
図 3-4 2 線式接続（ローカルセンシング） 3-5
図 3-5 4 線式接続（リモートセンシング） 3-6
図 3-6 3 端子デバイスに 2 台の SMU を接続（ローカルセンシング） 3-7

図 3-7 3 端子デバイスに 3 台の SMU を接続（ローカルセンシング） 3-8
図 3-8 高インピーダンスガード 3-9
図 3-9 ノイズシールド 3-9
図 3-10 危険電圧（>42V）に対応する安全シールド 3-10
図 3-11 図 3-10 に示す試験構成の接続 3-11
図 3-12 ノイズシールド、安全シールド、およびガードの接続 3-11
図 3-13 260x 型をフローティングにする 3-14
図 3-14 図 3-13 に示すフローティングの SMU 接続 3-14

4 基本操作

図 4-1 基本ソースメジャー構成 4-6
図 4-2 測定トリガシーケンス 4-10
図 4-3 2 線式抵抗センシング 4-19
図 4-4 4 線式抵抗センシング 4-20

5 スイープ操作

図 5-1 階段スイープのタイプ 5-3
図 5-2 線形階段スイープ 5-4
図 5-3 対数階段スイープ（1V から 10V、5 ステップ） 5-5
図 5-4 パルススイープの例 5-7
図 5-5 カスタムスイープの例 5-7

6 レンジ、桁数、速度、リラティブ、およびフィルタ

図 6-1 移動平均フィルタと反復フィルタ 6-14

8 ソースメジャーの概念

図 8-1 2 種類の基本スイープ波形 8-4
図 8-2 パルススイープの例 8-5
図 8-3 パワーを連続的に変化させたときの動作限界 8-7
図 8-4 電流ソース（I -Source）の限界 8-9
図 8-5 電流ソース（I -Source）の動作限界 8-11
図 8-6 電圧ソース（V-Source）の限界 8-13
図 8-7 電圧ソース（V -Source）の動作例 8-15
図 8-8 電流ソース構成 8-17
図 8-9 電圧ソース構成 8-18
図 8-10 測定専用の回路構成 8-19
図 8-11 ガード使用時とガード無し時測定の比較 8-22
図 8-12 パルス周期 8-23
図 8-13 パルスのライズ / フォールタイム 8-24

9 システムの拡張 (TSP-Link)

図 9-1 TSP-Link 接続 9-3

10 デジタル I/O とインターロック

図 10-1 デジタル I/O ポート 10-3
図 10-2 デジタル I/O ポートの回路構成 10-4
図 10-3 インターロックの使用法 10-9

11 通信インターフェイス

図 11-1 IEEE-488 コネクタ 11-3
図 11-2 IEEE-488 接続 11-4
図 11-3 IEEE-488 と RS-232 のコネクタ位置 11-5
図 11-4 RS-232 インターフェイス コネクタ 11-13

14 ディスプレイの操作

図 14-1 メッセージ表示の行 / 桁構成 14-6

15 性能確認

図 15-1 電圧確度確認のための接続 15-7
図 15-2 電流確度確認のための接続（100nA ～ 1A レンジ）..... 15-10
図 15-3 電流確度確認のための接続（3A レンジ）..... 15-11

16 校正

図 16-1 電圧校正のための接続 16-9
図 16-2 電流校正のための接続（100nA ～ 1A レンジ）..... 16-13
図 16-3 電流校正のための接続（3A、10A レンジ）..... 16-17

17 日常のメンテナンス

図 17-1 電源ヒューズの交換 17-2

D ステータスモデル

図 D-1 ステータスの概要 D-3
図 D-2 ステータスモデル（システムサマリと標準イベントレジスタ）..... D-4
図 D-3 ステータスモデル（動作イベントレジスタ）..... D-5
図 D-4 ステータスモデル（クエスチョナブルイベントレジスタ）..... D-6
図 D-5 ステータスモデル（測定イベントレジスタ）..... D-7
図 D-6 16-bit ステータスレジスタ D-10
図 D-7 ステータスバイトとサービス要求（SRQ）..... D-12
図 D-8 標準イベントレジスタ D-20

表リスト

1 ご使用前に

表 1-1 メインメニュー 1-19
表 1-2 設定メニュー 1-20
表 1-3 デフォルト設定 1-24

2 TSP プログラミングの基本

表 2-1 電圧をスイープし、電流を測定するスクリプト例 2-39
表 2-2 関数を使用するスクリプト例 2-40
表 2-3 スクリプトの中で対話操作を行う部分の例 2-41

3 DUT 試験の接続

表 3-1 前面パネルからセンスモードを選択 3-6
表 3-2 センスモードを選択するコマンド 3-7
表 3-3 出力 OFF 状態を選択するコマンド 3-16

4 基本操作

表 4-1 ソースメジャー機能 4-3
表 4-2 最大コンプライアンス値 4-4
表 4-3 コンプライアンスコマンド 4-5
表 4-4 オートゼロの設定項目 4-8
表 4-5 オートゼロコマンドとオプション 4-9
表 4-6 Trigger コマンド 4-12
表 4-7 基本的なソースメジャー コマンド 4-15

5 スイープ操作

表 5-1 対数スイープポイント 5-6
表 5-2 階段スイープ関数 5-9
表 5-3 パルススイープ関数 5-10
表 5-4 カスタムスイープ関数 5-10
表 5-5 スイープ例が使用するパラメータ 5-11

6 レンジ、桁数、速度、リラティブ、およびフィルタ

表 6-1 ソースと測定レンジ 6-2
表 6-2 レンジコマンド 6-6
表 6-3 桁数コマンド 6-8
表 6-4 速度コマンド 6-9
表 6-5 リラティブ コマンド 6-11
表 6-6 フィルタコマンド 6-15

7 バッファ（データ保存）

表 7-1 データ保存コマンド 7-5
表 7-2 バッファへの保存をコントロールする属性 7-7
表 7-3 バッファの読み出し専用属性 7-7
表 7-4 バッファをコントロールするプログラミング例 7-8
表 7-5 バッファの読み取り専用属性をコントロールするプログラミング例 7-8
表 7-6 呼び出し属性 7-9
表 7-7 バッファのステータスビット 7-10

8 ソースメジャーの概念

表 8-1 最大コンプライアンス値 8-3

9 システムの拡張 (TSP-Link)

表 9-1 装置の前面パネルからノード番号を割り付ける方法 9-4
表 9-2 前面パネルから TSP-Link をリセットする手順 9-5
表 9-3 TSP-Link リセットコマンド 9-5

10 デジタル I/O とインターロック

表 10-1 デジタル I/O ビット 10-5
表 10-2 デジタル I/O コマンド 10-7

11 通信インターフェイス

表 11-1 一般バスコマンド 11-7
表 11-2 RS-232 インターフェイス コマンド 11-11
表 11-3 RS-232 コネクタのピン配列 11-13
表 11-4 PC のシリアルポート ピン配列 11-14

14 ディスプレイの操作

表 14-1 関数 / 属性値とセクショントピックス対照表 14-2
表 14-2 ビット位置と表示ランプの対応 14-12
表 14-3 display.sendkey が送るキーコード一覧 14-16
表 14-4 display.getlastkey が返すキーコード値 14-17

15 性能確認

表 15-1 推奨試験装置 15-4
表 15-2 出力電圧の確度限界値 15-8
表 15-3 電圧測定の確度限界値 15-9
表 15-4 出力電流の確度限界値 15-10
表 15-5 電流測定の確度限界値 15-12

16 校正

表 16-1 校正に使用できる推奨試験装置 16-4
表 16-2 校正ステップ 16-6
表 16-3 校正コマンド 16-7

17 日常のメンテナンス

表 17-1 電源ヒューズ 17-3

B エラーメッセージとステータスメッセージ

表 B-1 エラーキュー コマンド B-2
表 B-2 エラー一覧 B-3

C 共通コマンド

表 C-1 共通コマンド C-2
表 C-2 スクリプトコマンドとの対応 C-3

D ステータスモデル

表 D-1 ステータス関数とレジスタ D-8
表 D-2 レジスタのリセット、キューのクリアに使用するコマンド D-9
表 D-3 ステータスバイトとサービス要求イネーブルレジスタコマンド D-15
表 D-4 システムノードと SRQ イネーブルレジスタビットに関連する属性値 D-16
表 D-5 システムサマリイベントに関連するコマンド D-17
表 D-6 標準イベントコマンド D-20
表 D-7 ステータスイベントステータスレジスタとビット D-21
表 D-8 動作イベントコマンド D-22
表 D-9 クエスチョナブルイベントコマンド D-24
表 D-10 測定イベントコマンド D-27
表 D-11 エラーキューコマンド D-30

1 ご使用前に

セクション 1 のトピックス

はじめに , 1-2
機能と特徴 , 1-2
本マニュアルのセクション構成 , 1-2

一般事項 , 1-3
保証について , 1-3
連絡先について , 1-3
安全に関するシンボルと用語 , 1-3
開梱と検査 , 1-4
オプションとアクセサリ , 1-5
ユーザマニュアル , 1-6
リファレンス マニュアル , 1-6

前面パネルと背面パネルに慣れましょう , 1-6
前面パネルのまとめ , 1-6
背面パネルのまとめ , 1-6

冷却用通気口 , 1-12

電源を入れる , 1-13
電源ラインの接続 , 1-13
起動シーケンス , 1-14
ブザー , 1-14

表示モード , 1-15

編集用コントロール要素 , 1-16
ソースとコンプライアンスの編集 , 1-16
メニュー ナビゲーション , 1-17

メニュー , 1-18
メインメニュー , 1-18
設定メニュー , 1-18

インターフェイスの選択 , 1-21
GPIB インターフェイス , 1-21
RS-232 インターフェイス , 1-21

エラー / ステータスメッセージ , 1-21

デフォルト設定 , 1-22
前面パネルの設定 , 1-22
リモート操作用のセットアップ , 1-23

リモートプログラミング , 1-25
読み込み要求 , 1-25
コマンド設定要求 , 1-25

はじめに

機能と特徴

- ± DC 電圧出力（1 μ V ～ 40.4V）
- ± DC 電流出力（1pA ～ 3.03A）
- ± DC 電圧測定（1 μ V ～ 40.4V）
- ± DC 電流測定（1pA ～ 3.03A）
- 抵抗および電力測定機能
- 独立した 2 系統のソースメータ チャンネル（2602 型のみ）
- 40W、4 象限の電流ソース / シンク能力（1 チャンネルあたり）
- あらゆるホスト インターフェイスからアクセス可能な TSP（Test Script Processor）が装置コントロールコマンドを含む試験用スクリプトに高速応答
- 複数のスイープ機能を持つスクリプトを標準添付（線形階段波、対数階段波、固定パルス、カスタムスイープ）
- 5 種類まで定義できるユーザスクリプト
- 100,000 個以上のソース測定値をバッファに保存 / 読み出し可能
- 読み取りノイズ低減に有効なフィルタ機能
- 2 種類のリモートインターフェイスをサポート：IEEE-488 (GPIB)、RS-232
- TSP 対応装置間での通信とトリガ送信を可能にする TSP- リンク
- デジタル I/O ポートを使用して 260x 型から他のデバイスをコントロール可能

本マニュアルのセクション構成

本マニュアルの PDF 版をご覧の場合はウィンドウの左側に表示される「ブックマーク」をクリックすることにより、該当するセクションを表示させることができます。このタブは同時に様々なセクションとそのトピックスへのリンクにもなっています。

マニュアルが含むセクションの一覧をご覧になりたい場合は、マニュアル初頭の「目次」を開いてください。

一般事項

保証について

保証に関する説明は本マニュアルの最初に記載されています。ご使用の 260x 型に保証規定に従ってサービスを実施する必要が生じた場合は、最寄のケースレー代理店または認定修理施設に詳細をお問い合わせください。修理品を返送される場合は、サービス書式（本マニュアルの巻末に添付）に必要事項を完全に記入のうえ、それを返送品に添付して修理担当施設に必要な情報をお伝えください。

連絡先について

本マニュアルの最初の部分に世界中の弊社連絡先電話番号を記載してあります。ご不明の点については最寄りのケースレー代理店、または直接弊社のアプリケーションエンジニア（888-Keithley (534-8453)）または 800-552-1115（米国、カナダのみ）までお問い合わせください。Web サイト（www.keithley.com）を使用してアプリケーションエンジニアに連絡することも可能です。

安全に関するシンボルと用語

装置本体および本マニュアルでは以下のシンボルと用語を使用します。

 シンボルが表記された個所については、マニュアルに記載の操作説明を必ずご覧ください。

 シンボルが表記された個所の端子には高電圧が印加されている可能性があります。このような電圧に直接触れないようにするために標準的な安全対策を実施してください。

 シンボルが表記された個所は高温になっている可能性があります。この部分に直接手を触れないでください。

本マニュアルの「**警告**」という見出しに続く文は、使用する人に傷害（場合によっては致命的な）を引き起こす可能性のある危険性について説明しています。該当する操作を実行する前に必ず説明の内容をよく読み、理解してください。

本マニュアルの「**注意**」という見出しに続く文は装置の損傷に結びつく危険事項を説明しています。このような損傷は保証規定を無効にすることがありますからご注意ください。

開梱と検査

損傷有無の検査

260x 型は厳重な電気的、機械的検査を受けて工場を出荷されます。輸送容器を開いて全品目を取り出したら、輸送中に物理的な損傷が発生した兆候が見られないかチェックしてください。（ディスプレイレンズ等には保護用のフィルムを被せていることがあります。これらのフィルムは取り除いてかまいません。）損傷が発見された場合は直ちに輸送業者に連絡してください。配送時の梱包カートンなどは将来の輸送に備えて保管しておいてください。260x 型をカバーから取り出すときは以下の取扱い上の注意をお守りください。

取扱い上の注意事項

- 260x 型を持ち上げるときはカバーまたはハンドルを掴んでください。
- 静電気防止バッグから 260x 型を取り出したら、目視で分るような物理的損傷の兆候がないかチェックしてください。このような損傷が発見された場合は直ちに輸送業者に連絡してください。
- 260x 型をすぐに設置して接続する必要のない場合は、ユニットを静電気防止バッグに収めた状態でオリジナル輸送容器に入れて保管してください。

オプションとアクセサリ

コネクタ

ソースメータ DUT インターフェイスコネクタキット – キットに含まれるフード付き端子コネクタは、ソースメータ側の測定端子に適合するように設計されています。ソースメータのチャンネル毎にキットが 1 つ添付されます。

GPIB ケーブルアダプタ（260x 型を GPIB バスに接続）

7006-1 型、7006-2 型 – シングルシールド式 GPIB ケーブル。ストレートコネクタ（非積み上げ式）と貫通接続スタイルのコネクタで終端されています。ケーブル長は Model 7006-1 が 1m、Model 7006-2 が 2m です。

7007-05、7007-1、7007-2、7007-4 型 – 二重シールドタイプの上級 GPIB ケーブル。高い性能を発揮して長寿命を実現するために両端を貫通式金属ハウジングで終端してあります。ケーブル長はそれぞれ、Model 7007-05 は 0.5m、Model 7007-1 は 1m、Model 7007-2 は 2m、Model 7007-4 は 4m です。

7010 型 – シールド付き IEEE - IEEE アダプタ。これを使用すると背面パネルと GPIB ケーブルコネクタ間のスペースが大きくなり、ケーブルやコネクタ類へのアクセスが容易になります。

RS-232 ケーブル（260x 型を RS-232 へ接続）

シールド付 RS-232 ケーブル 7009-5 型 – 260x 型の RS-232 コネクタを PC の RS-232 インターフェイスへ接続するストレートケーブル。ケーブル長 5ft。電磁干渉（EMI）防止のためシールド付きのケーブルとコネクタを使用しています。

TSP- リンク ケーブル（260x 型を TSP- リンクへ接続）

CA-180-3A CAT 5 ケーブル – 260x 型の TSP- リンクを他の装置の TSP- リンクへ接続するときに、この CAT5 LAN ケーブル（クロスケーブル）を使用します。

デジタル I/O ポート ケーブル（デジタル I/O から他のデバイスへの接続）

2600 型 -TLINK トリガケーブル – シリーズ 2600 装置のデジタル I/O ポートを TLINK（Trigger Link）を備えた他の Keithley 装置に接続します。

CA-126-1 DB-25 ケーブル – オスの DB-25 をメスの DB-25 へ接続するケーブル。長さ 1.5m（5ft）。デジタル I/O ポートを他の装置へ接続するときに使用します。

ユーザマニュアル

260x 型には印刷されたユーザマニュアルが標準で添付されています。また、製品情報 CD-ROM にも PDF ファイルとして書き込まれています。このマニュアルには装置の基本的な操作方法が説明されています。

リファレンス マニュアル

リファレンス マニュアルは CD-ROM に書かれた PDF ファイルとして提供されます。このマニュアルはユーザマニュアルが説明するトピックスへの追加情報を提供することに加えて、装置の更に高度な使用方法やメインテナンス情報を含んでいます。

前面パネルと背面パネルに慣れましょう

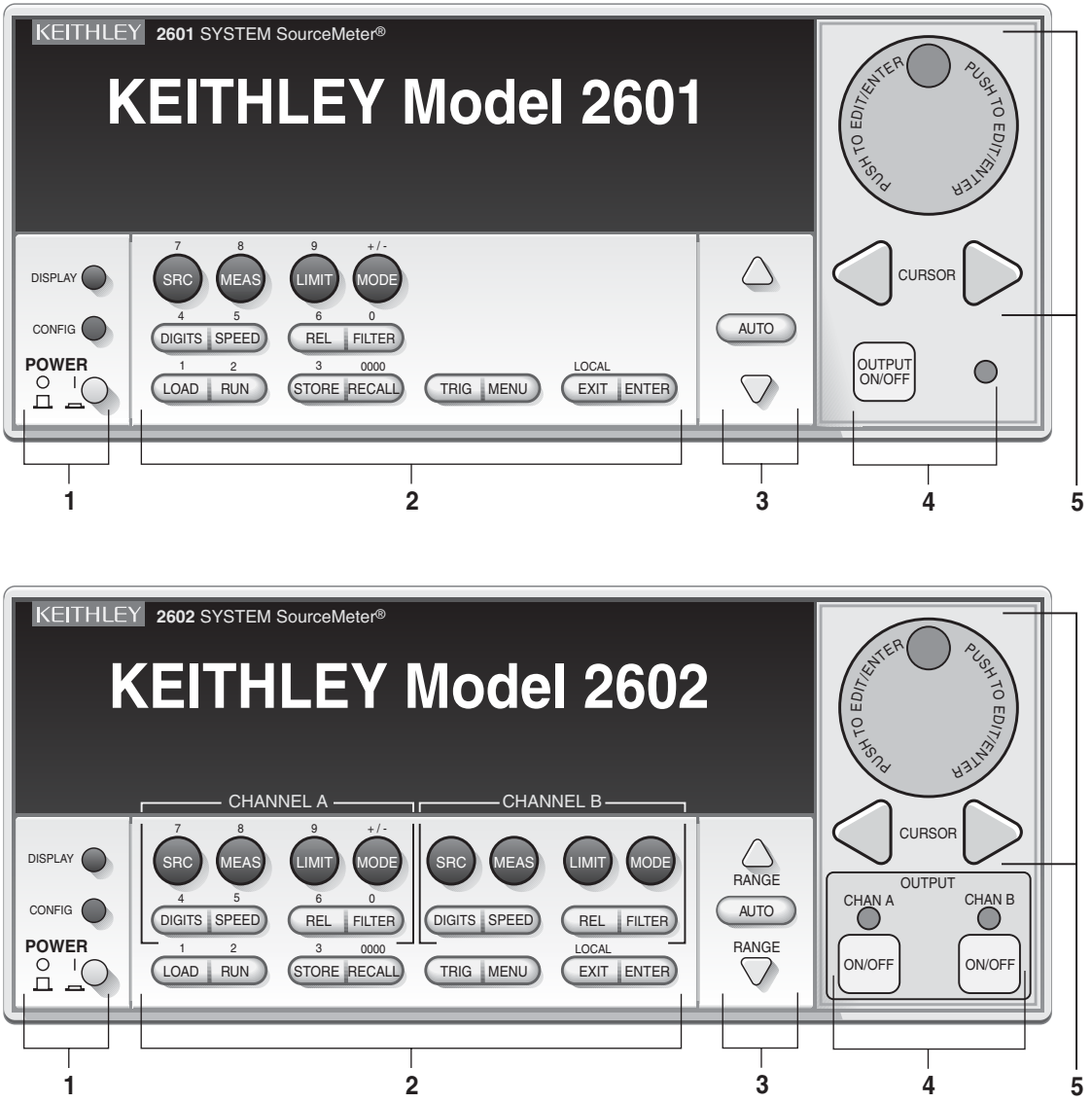
前面パネルのまとめ

2601、2602 型の前面パネルを図 1-1（p.1-7）に示します。前面パネル上のコントロール要素の説明が図 1-1 の後に続きます。

背面パネルのまとめ

2601、2602 型の背面パネルを図 1-2（p.1-10）に示します。背面パネル上のコントロール要素の説明が図 1-2 の後に続きます。

図 1-1
2601、2602 型の前面パネル



注	2601 型は 1 系統のソースメータチャンネル（Channel A）を持ち、2602 型は 2 系統のソースメータチャンネル（Channel A、B）を持っています。

1 特殊キーと電源スイッチ

DISPLAY	各種ソースメジャーの表示とユーザメッセージ表示モードを切換えます。また、2602 型ではシングル / デュアルチャンネル表示のいずれかを選択します。
CONFIG	機能や操作の設定を行うときに使用します。
POWER	電源スイッチです。“I” 側に倒すとソースメータに電源が入り、“O” 側に倒すと電源が切れます。
数字キー	EDIT モードで数字キー（0-9、+/-、0000）を使用することにより、数値を直接入力することができます。

2 ソース測定のセットアップ、性能コントロール、および特殊操作

上段 – ソース測定のセットアップ

2601、2602 型：

SRC	Channel A – ソース機能（V、A）を選択し、編集のためにカーソルをソース機能の領域へ移動します。
MEAS	Channel A – 測定機能（V、A、Ω、W）を順番に切換えてゆきます。
LIMIT	Channel A – コンプライアンスリミット値編集のためにカーソルを該当領域へ移動します。
MODE	Channel A – 測定機能（V、A、Ω、W）を直接選択します。

2602 型のみ：

SRC	Channel B – ソース機能（V、A）を選択し、カーソルをソース機能の領域へ移動します。
MEAS	Channel B – 測定機能（V、A、Ω、W）を順番に切換えてゆきます。
LIMIT	Channel B – コンプライアンスリミット値編集のためにカーソルを該当領域へ移動します。
MODE	Channel B – 測定機能（V、A、Ω、W）を直接選択します。

中段

2601、2602 型：

DIGITS	Channel A – 表示分解能を切換えます（4-1/2、5-1/2、または 6-1/2 桁）。
SPEED	Channel A – A/D 変換器の測定ウィンドウを調節して測定速度を設定します。
REL	Channel A – 読み取り値からベースライン値を引き算するための相対値を調節します。
FILTER	Channel A – 読み取り値のノイズ低減のためにデジタルフィルタの設定を調節します。

2602 型のみ：

DIGITS	Channel B – 表示分解能を切換えます（4-1/2、5-1/2、または 6-1/2 桁）。
SPEED	Channel B – A/D 変換器の測定ウィンドウを調節して測定速度を設定します。
REL	Channel B – 読み取り値からベースライン値を引き算するための相対値を調節します。
FILTER	Channel B – 読み取り値のノイズ低減のためにデジタルフィルタの設定を調節します。

下段

LOAD	スクリプト実行のために、工場出荷時に書き込まれている、またはユーザが定義したスクリプトを読み出します。
RUN	最近選択した工場設定スクリプト / ユーザ定義スクリプトを実行します。
STORE	読み取り値、ソース値、タイムスタンプを 2 つある内部バッファのいずれかに保存します（後で読み出し可能）。
RECALL	2 つのバッファのいずれかに保存されている読み取り値、ソース値、タイムスタンプを読み出します。
TRIG	読み込みトリガを発生させます。
MENU	メインメニューへアクセスします。メインメニューではセットアップの保存 / 呼び戻し、リモートインターフェイスや電源周波数の選択、自己試験、シリアル番号やブザーの調節などを行います。
EXIT	選択内容をキャンセルしてメニューを終了します。装置のリモートコントロールを解除する LOCAL キーとしても使用します。
ENTER	選択内容を確定させて次の項目へ移動するか、またはメニューを終了します。

3 レンジキー

△ 、▽	1 レベル上、または 1 レベル下のソース、測定レンジを選択します。
AUTO	ソースまたは測定レンジ自動選択機能の ON/OFF。

4 出力コントロールと LED ステータスランプ

OUTPUT ON/OFF	ソース出力の ON/OFF 切換え。
LED ランプ	出力が ON のときに点灯します。

5 回転ノブと CURSOR キー

ソースを編集するときは、CURSOR キーでカーソル位置を指定してからノブを回転させてソース値、コンプライアンス値を変更します。回転ノブはソース編集モードの ON/OFF にも使用します。

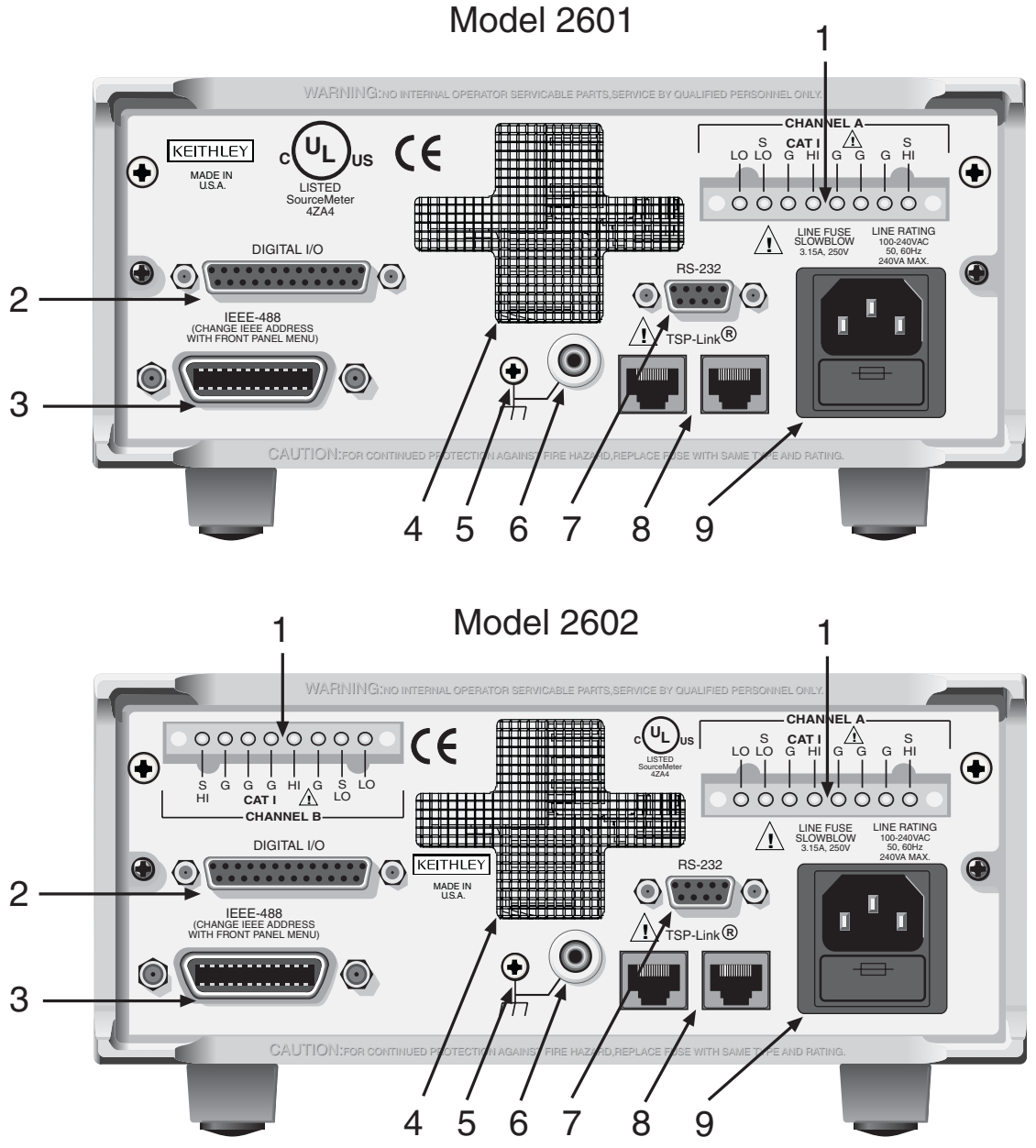
メニューに入っているときは、CURSOR キーまたは回転ノブを使用してメニュー項目上のカーソル位置をコントロールします。メニューで値を表示しているときは、CURSOR キーでカーソルの位置をコントロールし、ノブを回転させて値を変更します。メニュー項目を開く、あるいはメニューのオプションや値を選択するときはこのノブを押します。

6 表示ランプ（図には示されていません）

EDIT	ソース編集モードの単位を表示します。
ERR	読み取り値に問題があります、または計算ステップが無効です。
REM	リモートモードが使用する単位を表示します。
TALK	ユニットの TALK アドレス。
LSTN	ユニットの LISTEN アドレス。
SRQ	サービス要求。
REL	相対モード ON。
FILT	アナログフィルタまたは平均化フィルタ ON。
AUTO	ソースまたは測定レンジ自動選択機能 ON。
ARM	ユニットの準備が整い即実行可能。
TRIG	外部トリガが選択されています。

*(アスタリスク)読み込んだ値をバッファに保存中。

図 1-2
2601、2602 型の背面パネル



1 CHANNEL A、CHANNEL B（後者は 2602 のみに存在）

ソース、センス、ガード信号を接続する入 / 出力コネクタ。

2 DIGITAL I/O

DB-25（メス）コネクタ。デジタル入出力用として 14 本のピンを使用し、1 本を Output Enable に使用します。DB-25（オス）コネクタの付いたケーブルを使用してください（Keithley 部品番号：CA-126-1CA）。

3 IEEE-488

IEEE-488（GPIB）で動作させるためのコネクタです。7007-1 型、7007-2 型などのシールド付ケーブルを使用してください。

4 冷却用排気口

内蔵冷却ファンの排気口です。過熱防止のため、排気口周りに障害物を置かないでください。

5 筐体 GND

筐体を接地するための端子ネジです。

6 低ノイズ筐体 GND

Output HI または LO を筐体へ接続するための接地用ジャック。

7 RS-232

DB-9（メス）コネクタ。RS-232 で動作させる場合は、ストレートタイプ（ヌルモデム不可）のシールド付 DB-9 ケーブル（Keithley 7009-5 型）を使用して PC へ接続してください。

8 TSP-Link

260x 型と他の TSP 対応装置間でのトリガのやり取りと通信を可能にする拡張インターフェイス。カテゴリー 5e 以上のクロスオーバー LAN ケーブル（Keithley 部品番号 CA-180-3A）を使用してください。

9 電源モジュール

AC ラインの受けロコンセントと電源ヒューズが収められています。260x 型装置は電圧 100 ～ 240V、周波数 50/60Hz の電源で動作します。

冷却用通気口

260x 型は側面に吸気口、背面に排気口を備えています。ラック取付けの場合は熱を効果的に散逸するために少なくとも 1 つの側面の近くに障害物を置かないでください。カバー上面には絶対に液体（水、コーヒーなど）の入った容器を置かないでください。万一液体がこぼれると、通気口を通って装置内部に侵入して重大な損害を発生させます。

過剰な熱の発生は 260x 型の損傷または性能の劣化をもたらします。260x 型は周囲温度が 50° C を超えない環境で使用してください。

注意 **有害な熱の蓄積を防止して仕様に規定の性能を発揮させるために以下の注意事項をお守りください。**

- **背面パネルの排気口と少なくとも 1 方の側面吸気口周辺には障害物を置かないでください。例え部分的な閉塞であっても冷却効果が著しく損われます。**
- **260x 型の冷却用吸排気口や表面に向けて空気の流れ（加熱空気でなかったとしても）を起こす装置をそばに配置しないでください。余分な空気の流れは装置確度に悪影響を及ぼすことがあります。**
- **260x 型をラックに取付けるときは、少なくとも一方の側面で適切な空気の流れを確保できるようにしてください。充分な空気の流れが存在することで 260x 型の表面から約 1 インチ以内の空気温度が適切なレベルに保たれ、すべての動作条件において仕様に規定された範囲の性能を発揮します。**
- **ラック取付けの際に、大電力を消費する装置を 260x 型の近くに配置すると過剰な熱を発生させる恐れがあります。260x 型が仕様確度を発揮するためには装置表面の周囲温度が指定された範囲に収まっていなければなりません。ラック環境内の自然対流だけで十分な冷却を行うためには、最も高温になる装置（電源など）をラックの一番上に設置してください。一般にラックの下側ほど温度が下がりますから、260x 型のような精密装置はラックのできるだけ下側に配置してください。260x 型の下にスペースパネルを配置するのも空気の流れを確保するために有効です。**

電源を入れる

電源ラインの接続

以下に説明する手順に従って 260x 型を電源ラインに接続し、装置に電源を入れてください。本ソースメータは電圧が 100 ～ 240V、周波数 50/60Hz の電源で動作します。装置が電源電圧を自動検出しますから、電圧設定用のスイッチは存在しません。装置に適合した範囲の電圧が接続されていることを確認してください。

注意　**許容範囲外の電源電圧で装置を動作させると装置を傷める原因となるばかりでなく、保証規定を無効にすることがあります。**

1. 電源ケーブルを接続する前に、前面パネルの電源スイッチがオフ（O）になっていることを確認してください。
2. 製品に添付された電源コードのプラグ（メス）を背面パネルの AC リセプタクルに差し込みます。電源ケーブルの他方の端を接地端子付 AC コンセントに接続してください。

警告　**260x 型に標準添付される電源ケーブルには接地端子付コンセントに接続するための独立した接地導線が設けられています。これを正しく接続すれば、装置の筐体が電源ケーブルの接地導線を介して電源 GND ラインに接続されます。接地端子を持たないコンセントの使用は（場合によっては致命的な）感電事故を引き起こす可能性があります。**

3. 前面パネルの電源スイッチをオン側（I）に倒し、装置に電源を入れます。

電源周波数

260x 型は周波数 50Hz または 60Hz の電源で動作します。最高のノイズ特性を実現するためには、以下の手順にしたがって実際に使用する電源周波数に合わせてユニットを設定してください。

1. MENU キーを押します。
2. LINE-FREQ を選択し、ENTER または回転ノブを押します。
3. 50Hz または 60Hz のいずれかを選択し、ENTER または回転ノブを押します。
4. EXIT キーを押してメニューから抜け出します。

リモート設定を行う場合は localnode.linefreq コマンドを使用します。例えば、次のコマンドを実行すると周波数が 60Hz に設定されます。

```
localnode.linefreq = 60
```

ヒューズの交換

背面パネルのヒューズボックスは AC リセプタクルの下に配置されています（図 1-2 参照）。このヒューズの目的は装置の電源ライン入力を保護することです。電源電圧用ヒューズの交換方法については「電源ヒューズの交換」（p.17-2）を参照してください。

起動シーケンス

電源が投入されると 260x 型は ROM、NVRAM、および RAM の自己診断を行います。その際、短時間だけ全部の表示セグメントと表示ランプが点灯します。自己テスト中に故障が検出された場合は、エラーメッセージを短時間だけ表示してから表示ランプ “ERR” が点灯します。（エラーメッセージ一覧が付録 B にまとめられています。）

注　保証期間内の製品に問題が発生した場合は、修理のために装置をケースレーまで返送してください。

エラーが検出されなかった場合、260x 型は次の順序で立ち上がります。

- OUTPUT ランプと表示ピクセルが点灯して数秒経過すると、装置のモデル番号とファームウェアのバージョン番号、電源周波数設定が短時間だけ表示されます。
- ノードと GPIB アドレスが短時間だけ次の形式で表示されます。

  KEITHLEY MODEL 2602
  NODE = 1　　GPIB = 26
- 続いて、ノードとシリアルポートのパラメータ設定値が次の形式で短時間だけ表示されます。

  KEITHLEY MODEL 2602
  NODE = 1　　SERIAL = 9600,8,N,1,NONE

システムの識別

シリアル番号、ファームウェア番号、校正日付を表示させるにはメインメニューの SERIAL# の項を選択（MENU > GENERAL > SYSTEM-INFO の順に選択）してから希望に応じて FIRMWARE、SERIAL#、または CAL を指定します。

リモートプログラミングではクエリコマンド *IDN? を使用してシステム情報を読み出します。

ブザー

ブザー機能をアクティブにしておくと、以下の動作を外部に知らせるためにブザーが鳴動します。

- 前面パネルのキーを押すと短いブザー音が発生します。
- 回転ノブを回転または押したときにも短いブザー音が発生します。
- ソース側出力がオンになると長いブザー音が発生します。

ブザー機能を前面パネルから操作するときは、まず MENU > GENERAL > BEEPER の順に選択し、希望に応じて ENABLE または DISABLE を指定します。

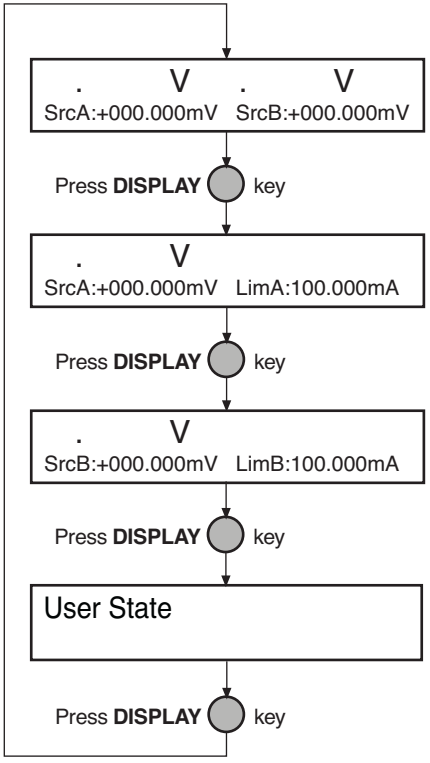
ブザー機能をリモートコントロールするには beeper.enable コマンドを使用します。例えば、次のコマンドを実行するとブザー機能がオンになります。

```
beeper.enable = 1
```

表示モード

DISPLAY キーを押すたびに図 1-3 に示す表示モードが順番に切り替わります。また、2602 型の場合には CHANNEL A（SMU A）と CHANNEL B（SMU B）のデュアル / シングルチャンネル表示の相互切換えにも DISPLAY キーを使用します。2601 型には 1 つのチャンネル（SMU A）しかありません。状態（User State）表示メッセージを定義するには特定の表示コマンドを使用します（表示メッセージについて詳しくはセクション 14 をご覧ください）。

図 1-3
表示モード



SMU A および SMU B のソースメジャー表示:
上の行には測定機能（V、A、Ω、W）が表示されます。
下の行にはソース機能（V、A）とレベルが表示されます。

SMU A のソースメジャーとコンプライアンス リミット表示:
上の行には測定機能（V、A、Ω、W）が表示されます。
下の行にはソース機能（V、A）とレベル、およびコンプライアンスリミット (A、V) が表示されます。

SMU B のソースメジャーとコンプライアンス リミット表示:
上の行には測定機能（V、A、Ω、W）が表示されます。
下の行にはソース機能（V、A）とレベル、およびコンプライアンスリミット (A、V) が表示されます。

ユーザが定義したメッセージおよびプロンプトが表示されます。

測定された読み取り値にまだトリガが掛かっていないことを示します。

編集用コントロール要素

ソースとコンプライアンスの編集

260x 型が編集モード（EDIT ランプ点灯）に入っているときに、編集用コントロール要素を使用してソースおよびコンプライアンスの値を設定します。ソース値の編集中はソース側の自動レンジ機能がオフになります。

ソース値の編集

1. ソース編集を行うために SRC キーを押します。ディスプレイのソース値フィールド上でカーソルが点滅表示を始めます。
2. CURSOR キーまたは回転ノブを使用して点滅カーソルを希望する数字の桁の上へ移動させます。
3. 回転ノブを押して編集モードに入ります（EDIT ランプが点灯します）。
4. 次の 2 通りの方法のいずれかを使用してソース値を変更します。
   - 回転ノブを回して選択されている桁の値を調節します。数値がその桁の最大値 / 最小値を超えると上位 / 下位の桁へ自動的にオーバーフローします。
   - 数字キー（0-9、+/-、0000）を使用して希望する値を直接入力します。‘+/-’は数値の符号を反転させ、‘0000’は値をゼロにセットします。
5. 希望する値が表示されたならば、ENTER キーまたは回転ノブを押して編集作業を完了させます。編集終了とともに EDIT ランプも消灯します。
6. 変更内容をキャンセルしてソース編集を終了したい場合は、ENTER または回転ノブを押すのではなく、EXIT キーを押してください。

コンプライアンス値の編集

1. 2601 型または 2602 型のシングルチャンネル表示モードの場合は、LIMIT キーを押してコンプライアンス値を直接変更します。
2. 2602 型のデュアルチャンネル表示モードの場合は、LIMIT または CONFIG を一度押してから、さらに LIMIT を使用してコンプライアンスリミット値を編集します。次に、VOLTAGE または CURRENT を選択します（CURSOR キーを押すか、または回転ノブを回して選択してください）。最後に ENTER キーまたは回転ノブを押してください。
3. CURSOR キーを使用するか、または回転ノブを回して点滅中のカーソルを希望する桁の上へ移動させます。
4. 回転ノブを押して編集モードに入ります（EDIT ランプが点灯します）。

5. 次の 2 通りの方法のどちらかを使用してコンプライアンス リミット値を変更します。
   - 回転ノブを回して選択されている桁の値を調節します。数値がその桁の最大値 / 最小値を超えると上位 / 下位の桁へ自動的にオーバーフローします。
   - 数字キー（0-9）を使用して希望する値を直接入力します。
6. 希望する値が表示されたならば、ENTER キーまたは回転ノブを押して編集作業を完了させます。編集終了とともに EDIT ランプも消灯します。
7. 変更内容をキャンセルして編集を終了するには EXIT キーを押してください。（メニューから脱出するにはこのキーを使用します。ENTER キーや回転ノブを押すのではないことに注意）。

メニュー ナビゲーション

260x 型が編集モードに入っていない（EDIT ランプ消灯）ときは、編集用のコントロール要素を使用してメインメニューと設定（Configuration）メニュー内を移動し、各種の選択や値の設定を行います（P.1-18 参照）。あるメニューに入ったならば、編集用のキーを次のように使用します。

メニュー項目の選択

1. CURSOR キーを使用するか、または回転ノブを回して点滅カーソルをメニュー項目の上へ移動させてその項目を選択します（またはサブメニューを開きます）。
2. ENTER キーまたは回転ノブを押して項目を選択します（またはサブメニューを開きます）。
3. 変更内容をキャンセル、またはメニュー構造から脱出したい場合は EXIT キーを押してください。

値の設定

値を変更する方法としては、現在の値を修正する方法と数値を直接入力する方法の 2 通りの方法があります。どちらも次の編集方法を用います。

- 値をゼロに設定するには数字入力キー “0000” を押します。
- 値の符号を反転させるには数字入力キー “+/-” を押します。

値を修正する方法

1. CURSOR キーを押すか、または回転ノブを回して点滅カーソルを変更の対象となる桁へ移動させます。
2. 回転ノブを押して編集モードに入ります（EDIT ランプ点灯）。
3. 回転ノブを回して値を設定します。現在の桁の最大値 / 最小値を超えるとカーソルが自動的に上位 / 下位の桁へ移動しますから、そのまま編集を続けることができます。
4. ENTER を押して値を確定させます。変更内容をキャンセルするには EXIT を押してください。

数値入力方法

1. CURSOR キーを押すか、または回転ノブを回して点滅カーソルを編集の対象となる数字の最上位桁へ移動させます。
2. 数字入力キー（0-9）を押して 1 つの桁に値を入力します。カーソルが右隣の桁へ移動します。
3. ステップ 2 の操作を必要回数繰り返して希望する数値を入力します。
4. ENTER を押して値を確定させます。変更内容をキャンセルするには EXIT を押してください。

メニュー

装置の操作に係わる多くの項目はメニューによって設定されます。メニューにはメイン（Main）メニューと設定（Configuration）メニューの 2 種類があり、前者は MENU キー、後者は CONFIG キーを使用してアクセスします。これらのメニューの内容を以下に説明します。メニューの使用法について詳しくは「メニュー ナビゲーション」（p.1-17）をご覧ください。

メインメニュー

メインメニューの内容と各メニュー項目に付随する参照項目をまとめて表 1-1（p.1-19）に示します。個々のメニュー項目にアクセスするには、まず MENU キーを押してから希望する項目を選択してください。

設定メニュー

設定メニューの内容と各メニュー項目に付随する参照項目をまとめて表 1-2（p.1-20）に示します。次の 2 通りの選択方法が用意されています。

- CONFIG を押し、続いて希望するサブメニューへ移行します。
- CONFIG を押し、続いて適切なキーを選択して押します。例：CONFIG キーに続いて REL キーを押すと直ちに相対（リラティブ）メニューに入ります。

表 1-1
メインメニュー

メニューの選択	説明	参照項目
SAVESETUP	ユーザセットアップ / 工場セットアップを保存 / 呼び出します。	p.1-22
SAVE	ユーザセットアップ（最高 5 種類）を保存します。	
RECALL	ユーザセットアップを呼び出します。	
POWERON	電源投入時のデフォルト設定を指定します。	
RESET	260x 型を工場設定の状態へリセットします。	
COMMUNICATION	インターフェイス、コントロール設定を選択します。	
INTERFACE_CFG	GPIB、RS-232 インターフェイスを設定します。	セクション 11
INTERFACE_SEL	GPIB または RS-232 インターフェイスのいずれかを選択、または AUTO を使用します。	セクション 11
TSPLINK_CFG	NODE と RESET TSP-Link を選択します。	セクション 9
TEST	ディスプレイの試験を実施します。	セクション 17
KEYS	キーを試験します。	
DISPLAY_PATTERNS	パターン表示試験を実施します。	
LINE-FREQ	電源周波数を指定します（50/60Hz）。	p.1-13
GENERAL	デジタル I/O、ブザー、システム情報をコントロールします。	
DIGOUT	デジタル I/O のビットを設定し、書き込み保護の有無を指定します。	セクション 10
BEEPER	ブザー機能の使用 / 不使用（ENABLE/DISABLE）を指定します。	p.1-14
SYSTEM-INFO	FIRMWARE、SERIAL#、CAL 日付を表示します。	p.1-14

表 1-2
設定メニュー

メニューの選択[1]	ショートカット[2]	説明	参照項目
CHANNEL-A		チャンネル A の設定：	
SRC	SRC	V- ソース検出、低レンジ、I- ソース低レンジ。	セクション 4
MEAS	MEAS	V および I の測定検出、低レンジ、オートゼロ。	セクション 4
LIMIT	LIMIT	V および I- ソースのコンプライアンス リミット。	セクション 4
SPEED	SPEED	測定速度（NPLC）。	セクション 6
REL	REL	相対値を設定します。	セクション 6
FILT	FILTER	デジタルフィルタをコントロールします。	セクション 6
OUTPUT	OUTPUT	オフ状態に設定し、デジタル I/O をコントロールします。	セクション 3、セクション 10
CHANNEL-B		チャンネル B の設定：	
SRC	SRC	V- ソース検出、低レンジ、I- ソース低レンジ。	セクション 4
MEAS	MEAS	V および I の測定検出、低レンジ、オートゼロ。	セクション 4
LIMIT	LIMIT	V および I- ソースのコンプライアンス リミット。	セクション 4
SPEED	SPEED	測定速度（NPLC）。	セクション 6
REL	REL	相対値を設定します。	セクション 6
FILT	FILTER	デジタルフィルタをコントロールします。	セクション 6
OUTPUT	OUTPUT	オフ状態に設定し、デジタル I/O をコントロールします。	セクション 3、セクション 10
COMMON		共通機能の設定：	
TRIG	TRIG	トリガ入力、カウント数、間隔、ディレイを設定します。	セクション 9
STORE	STORE	バッファのカウント数および書き込み先を設定します。	セクション 7

1. チャンネル B を備えているのは 2602 型のみです。
2. CONFIG キーに続いて該当キーを押すことにより直接メニューに入ることができます。

インターフェイスの選択

260x 型で選択できる基本インターフェイスの要点をまとめて以下に説明します。設定項目を含む詳しい内容についてはセクション 11 をご覧ください。インターフェイスを選択し、設定する手順については「メニュー ナビゲーション」（p.1-17）の項を参照してください。

GPIB インターフェイス

1. MENU を押してメインメニューを開きます。
2. COMMUNICATION を選択して ENTER を押します。
3. INTERFACE_SEL を選択して ENTER を押します。
4. GPIB を選択して ENTER を押します。
5. INTERFACE_CFG を選択して ENTER を押します。
6. GPIB を選択して ENTER を押します。
7. GPIB アドレス（0-30）をセットして ENTER を押します。
8. EXIT キーを押してメニューから抜け出します。

RS-232 インターフェイス

1. MENU を押してメインメニューを開きます。
2. COMMUNICATION を選択して ENTER を押します。
3. INTERFACE_SEL を選択して ENTER を押します。
4. RS-232 を選択して ENTER を押します。
5. INTERFACE_CFG を選択して ENTER を押します。
6. RS-232 を選択して ENTER を押します。
7. RS-232 インターフェイスのパラメータを設定します。
   - ボーレート（BAUD）を設定：300、600、1200、2400、4800、9600、19200、38400、57600 または 115200
   - ビット数（BITS）を設定：7 または 8
   - パリティ（PARITY）を設定：NONE、ODD、または EVEN
   - フローコントロール（FLOW-CTRL）を設定：NONE または HARDWARE
8. EXIT キーを押してメニューから抜け出します。

エラー / ステータスメッセージ

エラーメッセージとステータスメッセージは一時的にだけ表示されます。操作中、およびプログラミング中は前面パネルに何種類ものメッセージが表示されることがありますが、これらのメッセージは殆どがステータスまたはエラーメッセージ（付録 B の一覧表参照）です。

メッセージ（ステータス、エラーメッセージ両方）はキューに保持されています。このキューからエラーメッセージを取り出す方法については付録 D をご覧ください。

デフォルト設定

260x 型は 6 種類のセットアップ構成（ユーザが保存した 5 種類と工場設定）のいずれにでも簡単に戻すことができます。工場出荷時と同じように、260x 型は電源投入とともにオリジナルの工場設定の状態で立ち上がりますが、この設定内容をユーザ設定（5 種類）の位置へ保存することも可能です。オリジナルのデフォルト設定の内容を表 1-3（p.1-24）に示します。装置は起動時設定として保存してあるデフォルト設定に従って立ち上がります。

前面パネルの設定

ユーザセットアップを保存する

1. 保存したい動作モードに合わせて 260x 型を設定します。
2. MENU キーを押してメインメニューへアクセスします。
3. SAVE-SETUP を選択して ENTER を押します。
4. メニュー項目 SAVE を選択して ENTER を押します。
5. 希望するユーザメモリ位置（1-5）を選択して ENTER を押します。

セットアップを呼び戻す

1. MENU キーを押してメインメニューへアクセスします。
2. SAVE-SETUP を選択して ENTER を押します。
3. メニュー項目 RECALL を選択して ENTER を押します。
4. ユーザセットアップを呼び戻す場合は希望するメモリ位置（1-5）を選択し、工場デフォルトに戻したい場合は FACTORY を選択して、ENTER を押します。

起動時セットアップを指定する

1. MENU キーを押してメインメニューへアクセスします。
2. SAVE-SETUP を選択して ENTER を押します。
3. メニュー項目 POWERON を選択して ENTER を押します。
4. 工場デフォルトを指定する場合は FACTORY を選択し、5 種類のユーザセットアップのいずれかを指定するのであれば番号（1-5）を選択して、ENTER を押します。

リモート操作用のセットアップ

ユーザセットアップの保存と呼び出し

ユーザセットアップの保存と呼び戻しには setup.save と setup.recall コマンドを使用します。

```
setup.save(n)       --現在のセットアップをメモリに保存します
setup.recall(n)     --メモリに保存されているユーザセットアップを
                      呼び出します
                    --n = 1、2、3、4 または 5
```

デフォルトセットアップの呼び戻し

260x 型を工場で設定されたデフォルト状態に戻すには reset コマンドを実行します。

```
reset()             --すべてを工場デフォルトの状態に戻します
smua.reset()        --チャンネル A を工場デフォルトの状態に戻しま
                      す
smub.reset()        --チャンネル B を工場デフォルトの状態に戻しま
                      す（2602 型のみ）
```

起動時セットアップを選択する

起動時セットアップを選択するには setup.poweron コマンドを使用します。

```
setup.poweron = n       --起動時セットアップを選択します
                        --n = 0（*RST デフォルト）
                        --n = 1 -5（ユーザセットアップ 1-5）
```

表 1-3
デフォルト設定

設定項目 *	デフォルト値
A/D コントロール：	
オートゼロ	Auto
電源周波数	該当せず
ブザー	On
データ保存	該当せず
デジタル出力：	
出力値	該当せず
書き込み保護	該当せず
桁数	5-1/2
表示モード（2602 型）	2 チャンネル
フィルタ：	Off
平均化方式	反復
カウント数	1
GPIB アドレス	該当せず
リミット値：	
電流リミット	1A
電圧リミット	40V
測定：	
機能	電圧
電流測定レンジ	100mA
電圧測定レンジ	100mV
出力	Off
Off 状態	Normal
リラティブ	Off
電流値	0.0pA
電圧値	0mV
Ω 値	0mΩ
電力値	0mW
RS-232	該当せず
検出モード	2 線式

* 1 チャンネルに対してのみ意味を持つ項目以外は 2602 型の両方のチャンネルに共通に適用されます。

表 1-3 (続き)
デフォルト設定

設定項目 *	デフォルト値
ソース：	
機能	電圧
電流値	0A
電圧値	0V
電流レンジ	100nA
電圧レンジ	100mV
速度	Normal (1 PLC)
トリガ方式：	
トリガイン ソース	即時
カウント数	有限
インターバル	0s
ディレイ	0s
TSP- リンクのノード	1

* 1 チャンネルに対してのみ意味を持つ項目以外は 2602 型の両方のチャンネルに共通に適用されます。

リモートプログラミング

本マニュアルを通して、プログラミングのための情報を前面パネルの操作法と特に区別することなく説明します。基本的なコマンドに関する情報は表にまとめて説明します。プログラミングに固有な情報については、本マニュアルのセクション 2、12、および 13 をご覧ください。

読み込み要求

データ読み込みを要求するには、print コマンドの引数として適当な測定コマンドを与えます。例えば、次のコマンドはチャンネル A に対して電流値の読み込みを要求します。

```
print(smua.measure.i())
```

コマンド設定要求

同じように、コマンドに関する設定を要求する場合にも、該当するコマンドを print コマンドの引数として与えます。例えば、次のコマンドはチャンネル A に対して電圧ソースの設定を要求します。

```
print(smua.source.levelv)
```

2
TSP プログラミングの基本

セクション 2 のトピックス

はじめに , 2-2
プログラミングの概要 , 2-6
TSP ソフトウェアのインストール , 2-10
システム接続 , 2-10

Test Script Builder の使用 , 2-12
- Test Script Builder の起動 , 2-14
- 通信回線を開く , 2-15
- スクリプトの作成と変更 , 2-17
- スクリプト起動条件の設定 , 2-22
- スクリプトの起動 , 2-25
- TSP ファイルの実行 , 2-26
- 260x 型装置からスクリプトを読み出す , 2-26
- インスツルメンツコンソール（Instrument Console）, 2-28
- ファイル管理タスク , 2-34

コマンドと文の送信 , 2-38
- 電圧と電流の発生と測定 , 2-38
- デジタル I/O ポートの読み出しと書き込み , 2-38
- ユーザ定義メッセージの表示 , 2-38

ユーザスクリプト , 2-39
- スクリプト例 , 2-39
- ユーザスクリプトの作成 , 2-42
- ユーザスクリプトの保存 , 2-43
- ユーザスクリプトの実行 , 2-43
- ユーザスクリプトの変更 , 2-46
- スクリプトの管理 , 2-46

ファクトリスクリプト , 2-48
- ファクトリスクリプトの実行 , 2-48
- ファクトリスクリプトの変更 , 2-48
- 相違点：リモート vs. ローカル状態 , 2-49
- メモリに関する注意事項 , 2-51

TSL（Test Script Language）リファレンス , 2-52
- 予約語 , 2-52
- 変数と変数型 , 2-52
- 演算子 , 2-53
- テーブル / 配列 , 2-53
- 関数 , 2-54
- 優先度 , 2-55
- 論理演算子 , 2-55
- 連接 , 2-56
- 分岐 , 2-57
- ループ制御 , 2-58
- 標準ライブラリ , 2-59

はじめに

装置の標準的な使用法では、装置は送信されたコマンドに応答して動作します。コマンドメッセージには少なくとも 1 つのコマンドを含まれており、装置はコマンドが書かれている順番に従って応答します。

試験を実施するにあたって、コンピュータ（コントローラ）はコマンドシーケンスを装置側へ送信するようにプログラムされ、このコントローラが装置全体を組織的に協調させながら動作させます。通常、コントローラは装置側へ測定結果を要求するようにプログラムされており、それらの測定値をもとにコントローラが試験シーケンスの判定を行います。

TSP（Test Script Processor）をベースとする Keithley の装置は、コントローラが送信するコマンドメッセージに応答して標準的な測定装置として機能しますが、さらにその範囲をはるかに超える機能を実行することも可能です。

TSP (Test Script Processor)

スクリプト 組織的に動作シーケンスを実行します。
スクリプト言語 スクリプト作成に使用するプログラミング言語。

TSP（TSP）は装置内部でスクリプトを実行するエンジンであり、Lua (www.lua.org) と呼ばれるスクリプト言語で記述されたコードを実行する機能を備えています。以後 Lua を TSL（Test Script Language）とも呼ぶことにします。TSP は TSL コードファミリーを部分（正式にはチャンクと呼びます）ごとに実行します。装置に送られるメッセージの大部分は TSL チャンクとしてそのまま TSP により実行されます。装置へ送られる最も単純なメッセージは個々の装置コントロールコマンドです。これらのメッセージは TSP チャンクとして実行されますが、その使用法は標準的な装置の操作法と何ら変わるものではなく、単にユーザが送信したコマンドを装置が実行するだけです。したがって、コマンドメッセージを個別に送信している範囲では TSP がメッセージをチャックとして実行するかどうかは問題ではありません。

装置コントロールコマンドは TSP 内のライブラリとして実装されています。ある TSP 対応装置には使用できるコマンドセット全体が ICL（Instrument Control Library）として提供されます。したがって、TSP 対応装置はそれぞれが自分用の ICL を持っています。TSP 対応装置はすべて同じ TSL を実行しますが、一般に ICL は装置ごとに異なりますから装置ごとに応答できるコマンドも異なる可能性があります。

ICL コマンドは標準的な装置に送信されるコマンドに非常によく似ており、関数呼び出しや代入文のような形態をとります。例えば、チャンネル A の出力レベルを 1V にセットするコマンドは smua.source.levelv = 1 と表現され、チャンネル A 出力をオンに設定するコマンドは smua.source.output = smua.OUTPUT_ON と表現されます。これらのメッセージを独立に個別メッセージとして送信すると、それ自体が TSL チャンクとして処理されます。

コマンド 1 つ 1 つを個別のメッセージとして送信する必要はありません。例えば、上に説明した 2 つのコマンドを 1 メッセージ（1 つのチャンク）として送信することも可能です。その場合は 2 つのコマンドをスペース記号で区切って連接してください。連接後のチャンクは次のように表現されます。

```
smua.source.levelv = 1 smua.source.output = smua.OUTPUT_ON
```

ランタイム環境

すべてのスクリプト環境に共通する特徴はランタイム環境を備えていることです。TSP の場合には単にグローバル変数の集合体がランタイム環境となります。グローバル変数を使用すれば、装置に電源が入っている間ある値を記憶させておくことができ、その変数に新しい値が割り付けられることはありません。例えば、コマンド x = smua.measure.v() は電圧を測定して、その結果を “x.” という名前のグローバル変数に書き込むという指示を装置に与えます。

環境からグローバル変数を消去したい場合はその変数に値 nil を代入します。例えば、コマンド “x = nil” を実行することによりグローバル変数 x がランタイム環境から削除されます。装置の電源が切れると、ランタイム環境全体が失われます。ただし、SMU 不揮発性読み込みバッファの内容は失われません。

クエリ

TSP 対応の装置は本質的にクエリコマンドを持ちません。他のスクリプト環境と同じように、出力が必要な場合は print コマンドとその変形コマンドを使用します。print コマンドは 1 つの応答メッセージを生成します。

次に示すのは 2 つのコマンドから形成されるチャンクを使用して出力メッセージを生成させる例です。ここでは、測定を 1 回実行して値を返します。

```
x = smua.measure.v() print(x)
```

測定値は最初のコマンドが実行された段階でグローバル変数 x に保存されることに注意してください。

スクリプト

より複雑なコマンド シーケンスを TSP を使用して実行しようとする場合、シーケンス全体を単一のメッセージとして送ろうとすると操作が非常に煩雑になることがあります。特にループや分岐処理などの高度なスクリプト機能を利用するシーケンスの場合はこれが顕著になります。このため、コマンドメッセージのシーケンスを 1 つのチャンクにまとめるための特殊なメッセージが 2 種類用意されています。

その 1 つである loadscript メッセージを使用すると、TSP 対応の装置にそれに続く複数のメッセージを（即時実行するのではなく）まとめて読み込むように指示します。コマンドメッセージのシーケンスがすべて送信されたならば、endscript メッセージを使用して TSP 対応装置にその試験スクリプトのコンパイルを指示し、後続のメッセージで実行できるようにします。このチャンクは「アクティブ スクリプト」と呼ばれます。

script.run() を送信することによりアクティブ スクリプトを任意のタイミングで実行することができます。アクティブ スクリプトは script.run() コマンドが送信されるたびに何回でも実行可能であり、その都度再送する必要はありません。

loadscript と endscript メッセージを使用して新しいスクリプトを送信すると、TSP に対応した装置は現在のアクティブ スクリプトを新しいスクリプトで置き換えます。この方法でスクリプトを作成して使用できるということは、TSP 対応装置が備える非常に強力な機能の 1 つですが、ある一時にアクセスできるスクリプトは 1 つだけに限られるという制約があります。装置に同時に複数のスクリプトを保存できる「名前付きスクリプト」の使用法について次のトピックスで説明します。

名前付きスクリプト

名前付きスクリプトを作成するときにも loadscript を使用します。loadscript メッセージを使用して名前付きスクリプトを作成すると、アクティブ スクリプトを名前付きスクリプトで置き換える代わりに、ランタイム環境にグローバル変数が作成されてそこにスクリプトが保存されます。スクリプトがグローバル変数に保存されることから、スクリプトの名前は TSL 変数名として正しい名前である必要があります。

loadscript メッセージでスクリプトの名前を指定するには、loadscript キーワードの後に、スペース文字で区切って名前を付け加えます。例えば、loadscript MyScript というメッセージを受信した TSP 対応装置はスクリプト作成のためのコマンドメッセージを受け取り、それらのコマンドメッセージから MyScript という名前のスクリプトを作成します。スクリプトの最後であることを示すため、コマンドメッセージ送信の最後にやはり endscript メッセージを使用します。装置は endscript メッセージを受信するとスクリプトのコンパイルを開始します。エラーが検出されなければ、loadscript MyScript メッセージで指定された名前（MyScript）のグローバル変数が作成され、そこに即時実行可能なスクリプトが保存されます。名前付きスクリプトが装置へ成功裏に送信されると、それ以後は MyScript() を送信すれば何時でもスクリプトを実行することができます。もちろん、スクリプトに別な名前を付けた場合は実行コマンドにも同じ名前を使用します。

既存のスクリプトと同じ名前の新しいスクリプトを送信すると、新しいスクリプトによって旧いスクリプトが置き換えられます。既存のスクリプトのどれとも異なる名前で新しいスクリプトを送信すれば、送信済みスクリプトが削除されることはありません。名前付きスクリプトを使用すれば、ランタイム環境が使用できるメモリ容量の制限内であれば、任意の数のスクリプトを同時に使用することができます。

名前付きスクリプトはランタイム環境内にグローバル変数として保存されます。したがって、他のグローバル変数と同様にユニットの電源が切れると名前付きスクリプトも失われます。ただし、装置の不揮発性メモリを使用して電源が切れてもダウンロードされたスクリプトを消さずに保存することも可能です。詳しくは「ユーザスクリプトの保存」（p.2-43）をご覧ください。

関数

これまでに説明したように、名前付きスクリプトは TSL の関数と同じように振舞います。スクリプトと同じ名前の関数を呼び出すのと同じ方法で名前付きスクリプトを実行することができます。これも関数と同じように、スクリプトに値を返させることも可能です。ただし、パラメータを全く必要としないスクリプトも存在するという点で関数とは異なります。チャンクにパラメータを引渡すためには TSL 関数を作らなければなりません。

関数は次の形式のいずれかを使用してメッセージから作成します。

```
MyFunction = function （パラメータ 1、パラメータ 2） 関数本体終わり
```

または

```
function MyFunction( パラメータ 1、パラメータ 2） 関数本体終わり
```

ここに、関数本体とはその関数が呼び出されたときに実行される TSP チャンクを指します。上記の関数を実行するには次のメッセージを送信します。

```
MyFunction( パラメータ 1 の値、パラメータ 2 の値 )
```

ここに、パラメータ N の値とは関数呼び出し時に該当するパラメータに引き渡される値です。関数を定義すると、その関数もランタイム環境内のグローバル変数となることに注意してください。すべてのグローバル変数と同様に、関数も削除 / 上書きされるか、または電源が切れるまでランタイム環境に存在し続けます。

関数を生成するスクリプト

ほとんどの場合、1 つのメッセージだけで関数を定義しようとするのは非常に不便です。その解決策として、関数を定義するスクリプトを作成することができます。このタイプのスクリプトも他のスクリプトと良く似ていますが、スクリプトが実行されるまでは装置側に対して何の操作も行いません。関数を作成することは関数として機能するグローバル変数を作ることであったことを思い出してください。このグローバル変数は関数を生成するチャンクが実行されるまで存在しませんが、このケースではスクリプトがこのチャンクとして機能します。したがって、関数を生成するスクリプトが実行されるまでその関数は存在しません。初心者には混乱し易い点ですのでご注意ください。

例：MakeMyFunction という名前のスクリプトで関数 MyFunction を作成します。これを実行するメッセージ シーケンスを次に示します。

```
loadscript MakeMyFunction
    MyFunction = function (who)
        print("Hello " .. who)
    end
endscript
```

このメッセージ シーケンスが送信されると、MakeMyFunction スクリプトの内容は装置上のグローバル変数 MakeMyFunction に保存されます。しかし、その時点では MakeMyFunction スクリプトは実行されていませんから、関数 MyFunction はまだ存在しません。MakeMyFunction() メッセージが送信されると装置が MakeMyFunction スクリプトを実行し、それによって関数として機能するグローバル変数 MyFunction が作り出されます。

ここまでの処理が終わってから MyFunction("world") メッセージを送信すれば、装置が関数 MyFunction を実行して応答メッセージ “Hello world” を表示します。

プログラミングの概要

チャンクとは何か？

チャンクは 1 つ、または複数のプログラム文並びから構成される塊であり、プログラム文は逐次実行されます。チャンクにはスクリプトとして記述されるタイプと、そうでないタイプがあります。

単文チャンク – 次のプログラム文はそれ自体で 1 つのチャンクです。

```
print ("This is a chunk")
```

上記のチャンクを実行すると次の文字列を返します。

```
This is a chunk
```

マルチ文チャンク – チャンクは複数の命令文を含むことができます。1 つの行に複数の命令文を記述する場合はスペース記号で区切ります。例えば、次のチャンクは 2 つの文を含んでいます。

```
print ("This is a chunk") print ("that has two statements")
```

上記のチャンクを実行すると、2 つの文が逐次実行されて以下の文字列を返します。

```
This is a chunk
that has two statements
```

マルチチャンク – 次の 2 つの行のコードはチャンクとしては 2 つになります。最初のチャンクは SMU A のソースレベルを 1V に設定し、2 番目のチャンクが出力をオンにします。

```
smua.source.levelv = 1
smua.source.output = smua.OUTPUT_ON
```

スクリプト チャンク – スクリプト環境では試験プログラムコードのリスト全体が 1 つのチャンクになります。上に示した例の 2 つの文がスクリプトとして作成されたのであれば、2 つのコード行全体が 1 つのチャンクと見なされます。詳しくは次の「スクリプトとは何か？」をご覧ください。

スクリプトとは何か？

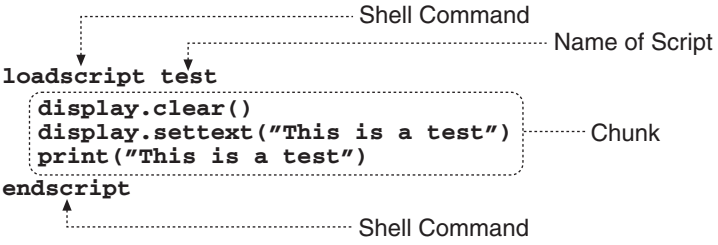
260x 型は TSP（Test Script Processor）を使用してスクリプトと呼ばれるプログラムを処理し、実行します。スクリプトとは装置コントロール コマンドとプログラム文の集合体です。スクリプトを “test” という名前で作成し、それをローディングする例を図 2-1 に示します。このスクリプトを実行すると、260x 型のディスプレイに “This is a test” メッセージが表示され、同じメッセージが PC へ送信されます。

例が示すように、スクリプトはプログラム コードのチャンクをシェルコマンドで囲んで作られます。図 2-1 の最初のシェルコマンドが “test” という名前のスクリプトを読み込み、最後のシェルコマンドがスクリプトの終了を知らせます。

図 2-1 に示すチャンクは 3 行のコードから構成されています。このチャンクが実行されると試験用メッセージが送信されて表示されます。このチャンクを実行させるコマンドは test() です。

注 スクリプトを実行するという表現を用いるのが通常ですが、実際に実行されるのはスクリプトに含まれるチャンクの部分です。

図 2-1
スクリプトの例



スクリプトは 260x 型ソースメータにローティングされて、そこで実行されます。ソースメータ側でスクリプトを実行する方が PC から試験プログラムを実行するよりもはるかに高速ですから、スクリプトを利用すれば PC からソースメータへ断片的にコマンドを送信する方法を使用せずに済みます。

プログラム文は変数や関数、分岐、ループ等の制御機能を使用できますから、これらの機能を利用してスクリプトの実行をコントロールすることができます。スクリプトはプログラムですから、プログラム言語で記述されます。ここで使用する言語は TSL（Test Script Language）と呼ばれます。TSL はスクリプト言語 Lua から派生した言語です。詳しくは「TSL（Test Script Language）リファレンス」（p.2-52）をご覧ください。

スクリプトにはファクトリスクリプトとユーザスクリプトの 2 種類があります。ファクトリスクリプトは Keithley 自体によって作成されて 260x 型の不揮発性 RAM に書き込まれたスクリプトです。ユーザスクリプトはユーザ自身のプログラム、または製品と共に提供される統合開発環境（IDE）ソフトウェアツールである Test Script Builder を使用して作成されます（「Test Script Builder の使用法」（p.2-12）参照）。260x 型ソースメータへ読み込み後のユーザスクリプトは不揮発性メモリへ保存することも可能です。

ランタイム環境

ランタイム環境とはユーザが作成したグローバル変数（スクリプト）の集合体です。スクリプトはランタイム環境に組み込まれることによって、初めてその実行と管理が可能となります。スクリプトは次の方法でランタイム環境に配置されます。

- 260x 型の不揮発性メモリに保存されたスクリプトは、装置に電源が入ると同時にランタイム環境内へ呼び出されます。
- ユーザが作成してローディングした名前付きスクリプトもランタイム環境に組み込まれます。
- 名前を持たないスクリプトもユーザが作成してローティングすることによりランタイム環境に組み込まれます。ただしランタイム環境に存在できる名前のないスクリプト（アクティブ スクリプトと呼びます）は 1 種類に限定されることに注意してください。もし名前のないスクリプトを別に作

成してローディングすると、ランタイム環境内にすでに存在する名無しスクリプトは新しいスクリプトによって上書きされます。

不揮発性メモリ

新規または変更後のスクリプトを 260x 型にローディングすると、スクリプトはランタイム環境に保存されますから装置の電源が切れると失われます。電源を切った後もスクリプトが消えないようにするには、不揮発性メモリに保存しなければなりません。このようにしておけば、260x 型に電源を再投入すると同時に、保存されていたすべてのスクリプトが「ランタイム環境」にローディングされます。

260x 型のランタイム環境と不揮発性メモリは明確に区別して考えなければなりません。ランタイム環境のスクリプトに変更を加えたとしても、不揮発性メモリに保存されている方のスクリプトはそのままで変更されません。変更後のスクリプトを保存することによって不揮発性メモリ上の旧バージョンが上書きされます。

TSP のプログラミングレベル

試験システム内のソースメータのプログラミングとコントロールには ICL（Instrument Control Library）コマンドと TSL プログラム文を使用します。このプログラミングには次の 3 段階のレベルがあります。

- コマンドと文の送信（p.2-38）– スクリプト化されていないチャンクが 1 行ごとに PC によって実行されます。
- ユーザスクリプト（p.2-39）– プログラムスクリプトを作成し、それを 260x 型ソースメータへローディングして装置側で実行します。
- 対話スクリプト（p.2-40）– このタイプのスクリプトはオペレータとの対話操作を通して実行されます。具体的にはユーザが定義したメッセージをソースメータのディスプレイに表示し、オペレータがそれに対応するパラメータ値を入力することによって処理が進行します。

スクリプトのプログラムモデル

スクリプトで使用する基本的なプログラミングのモデルを図 2-1 に示します。Keithley が工場で書き込むファクトリスクリプトは常に 260x 型の不揮発性メモリに保存され、ユーザが作成するスクリプトは指定によって不揮発性メモリに保存することも可能です。

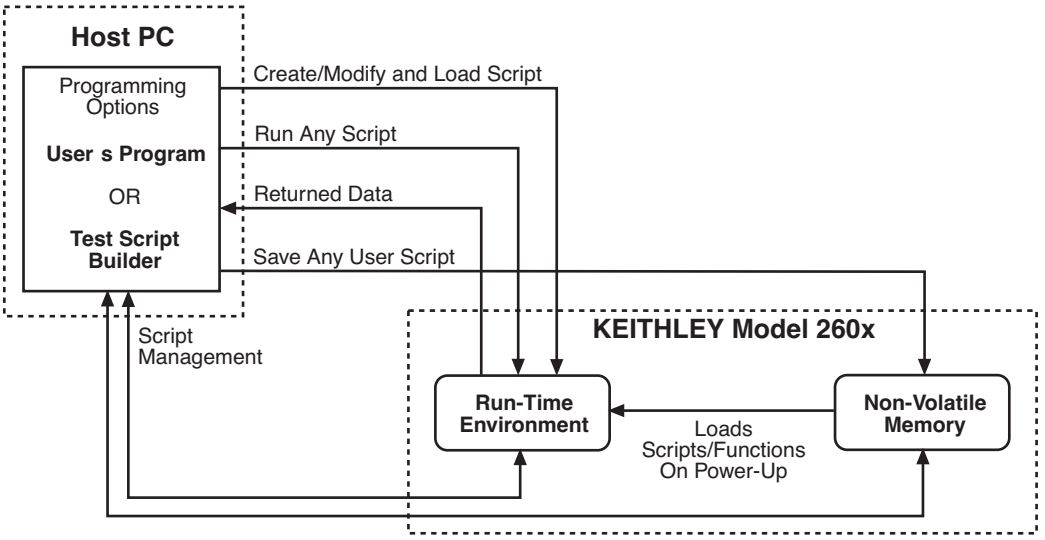
260x 型の電源が入ると、不揮発性メモリに保存されているすべてのスクリプト（ファクトリスクリプトとユーザスクリプト）がランタイム環境に呼び戻されます。自動実行するように設定されたユーザスクリプトがある場合は、全部のスクリプトのローディング終了後にそれが起動します。ランタイム環境中の任意のスクリプトは Test Script Builder またはユーザ自身のプログラムから実行が可能であり、得られた試験データ（例：読み取り値）が 260x 型から PC へ返されます。

ユーザスクリプトの作成にも Test Script Builder またはユーザ自身のプログラムを使用します。一旦ランタイム環境にローディングされたユーザスクリプトはいつでも実行可能な状態にあります。ランタイム環境中のユーザスクリプトは 260x 型の電源が切れると共に失われますが、不揮発性メモリに保存しておけば電源が切れても消えることはありません。

スクリプト管理のために次の機能を持つコマンドが用意されています。

- 不揮発性メモリに保存されているスクリプトを呼び戻します（例えば変更のため）。
- 不揮発性メモリ中のユーザスクリプトを消去します。
- 不揮発性メモリからスクリプトを呼び出してランタイム環境のスクリプトを復元します。

図 2-2
スクリプトのプログラミングモデル



TSP ソフトウェアのインストール

TSP ソフトウェアをインストールするときは、まず全てのプログラムを終了してから CD-ROM ドライブへ CD を挿入し、画面に表示される指示に従って操作します。ソフトウェアのインストールが自動的に始まらない場合は 「スタート」>「ファイル名を指定して実行」をクリックし、参照ボタンを押して CD ー ROM を開き、setup.exe をダブルクリックします。

システム接続

1 つの試験システムで最高 64 台までの 260x 型装置を使用することができます。試験システムのホストインターフェイスとして使用できるのは GPIB または RS-232 です。GPIB の場合は IEEE-488 ケーブルを使用して 260x 型装置のいずれかの 1 台を PC へ接続します。RS-232 の場合は DB-9 コネクタの付いたストレートケーブルを使用して 260x 型装置のいずれかの 1 台を PC へ接続します。

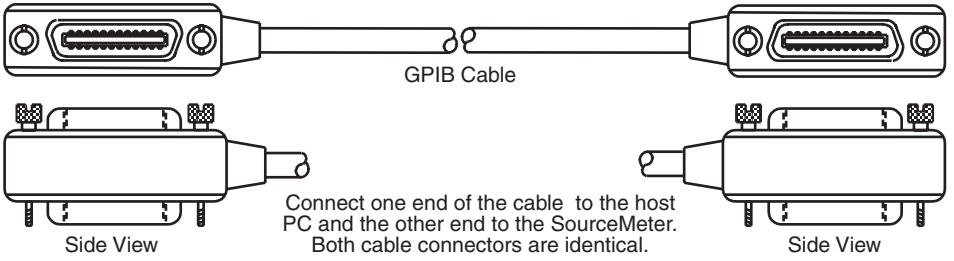
GPIB または RS-232 ケーブルを使用して PC へ接続するのは 1 台の 260x 型だけであることに注意してください。他の 260x 型装置との通信は TSP- リンク（セクション 9 参照）を介して行われます。

GPIB

GPIB 接続

260x 型を GPIB バスへ接続する際には図 2-3 に示す標準 IEEE-488 コネクタ付きのケーブルを使用してください。ソースメータ側の IEEE-488 コネクタは背面パネルに実装されています。ケーブルを接続するときは拘束ネジでしっかりと固定してください。

図 2-3
GPIB ケーブル



注　電磁輻射による干渉を最小限に抑えるため、必ずシールド付きの GPIB ケーブルを使用してください。ケースレーが提供するシールド付きケーブルの製品名は 7006 および 7007 型です。

GPIB ケーブルは積み上げ接続可能な構造になっています。260x 型以外の GPIB 装置をシステムへ追加接続したい場合は、GPIB ケーブルを 1 台の装置から次の装置へ芋づる式に延長してゆきます。

GPIB アドレス

GPIB アドレスは工場出荷時に 26 に設定されています。このアドレスは 0 から 30 までの範囲ならば任意の値を選択できますが、同じシステム内の他の装置に割り付けられた番号と重複することは許されません。

GPIB アドレスは通信メニューを使用して変更することができます。このメニューへアクセスするには、まず MENU キーを押してから COMMUNICATIONS を選択し、最後に GPIB を指定します。

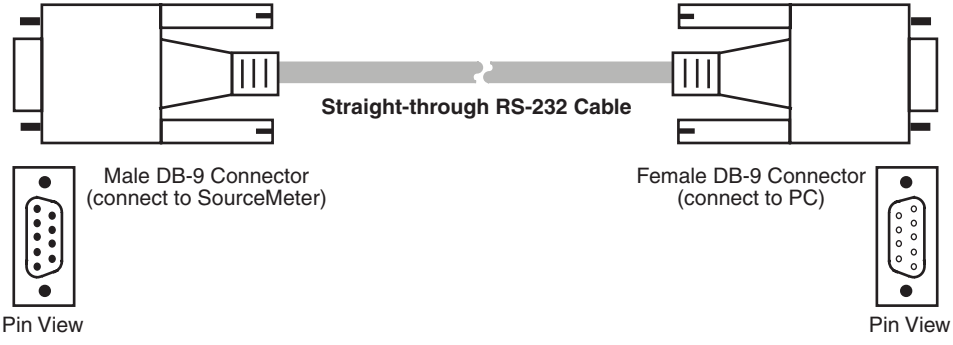
GPIB アドレスは不揮発性メモリに保存されます。したがって、電源の入り切りを繰り返しても、リセットコマンド（reset または *RST）を実行してもアドレス値が変化することはありません。

RS-232

RS-232 接続

RS-232 インターフェイスを使用して 260x 型を接続する場合はストレートタイプ（図 2-4）の RS-232 ケーブルを使用してください。ソースメータ側の RS-232 コネクタの位置を図 1-3 に示します。ケーブルを接続するときは拘束ネジでしっかりと固定してください。

図 2-4
RS-232 ケーブル（ストレート）



RS-232 の設定

RS-232 パラメータは工場出荷時次のように設定されています。

- ボーレート：9600
- データビット：8
- パリティ：None
- フロー制御：None

RS-232 の設定は通信メニューを使用して変更することができます。このメニューへアクセスするには、まず MENU キーを押してから COMMUNICATIONS を選択し、最後に RS-232 を指定します。

RS-232 の設定は不揮発性メモリに保存されます。したがって、電源の入り切りを繰り返しても、リセットコマンド（reset または *RST）を実行しても設定内容が変更されることはありません。

Test Script Builder の使用

Test Script Builder は製品と共に提供されるソフトウェアツールであり、以下の操作の実行に使用されます。

- ICL コマンドと TSL 文の送信
- コマンドとスクリプトに対する応答（データ）の受信
- ファクトリスクリプトの実行
- ユーザスクリプトの作成と実行

Test Script Builder の画面例を図 2-5 に示します。図から分るとおり、ワークスペースは 3 種類のウィンドウペインに分割されています。

プロジェクトナビゲータ

ワークスペースの左側ペインにあるのがプロジェクト ナビゲータです。このナビゲータは作成済みのプロジェクトフォルダとそれぞれのプロジェクト用に作成されたスクリプトファイル（.tsp）から構成されています。各プロジェクトフォルダに 1 つ以上のスクリプトファイルが保存されます。

図 2-5 に示すナビゲータには 2 つのプロジェクト (“BeeperTes” と “SourceMeasure”) が登録されており、図から分るとおり “BeeperTest” にはスクリプトファイルが 1 つ、“SourceMeasure” プロジェクトには 3 つのスクリプトファイルが含まれています。

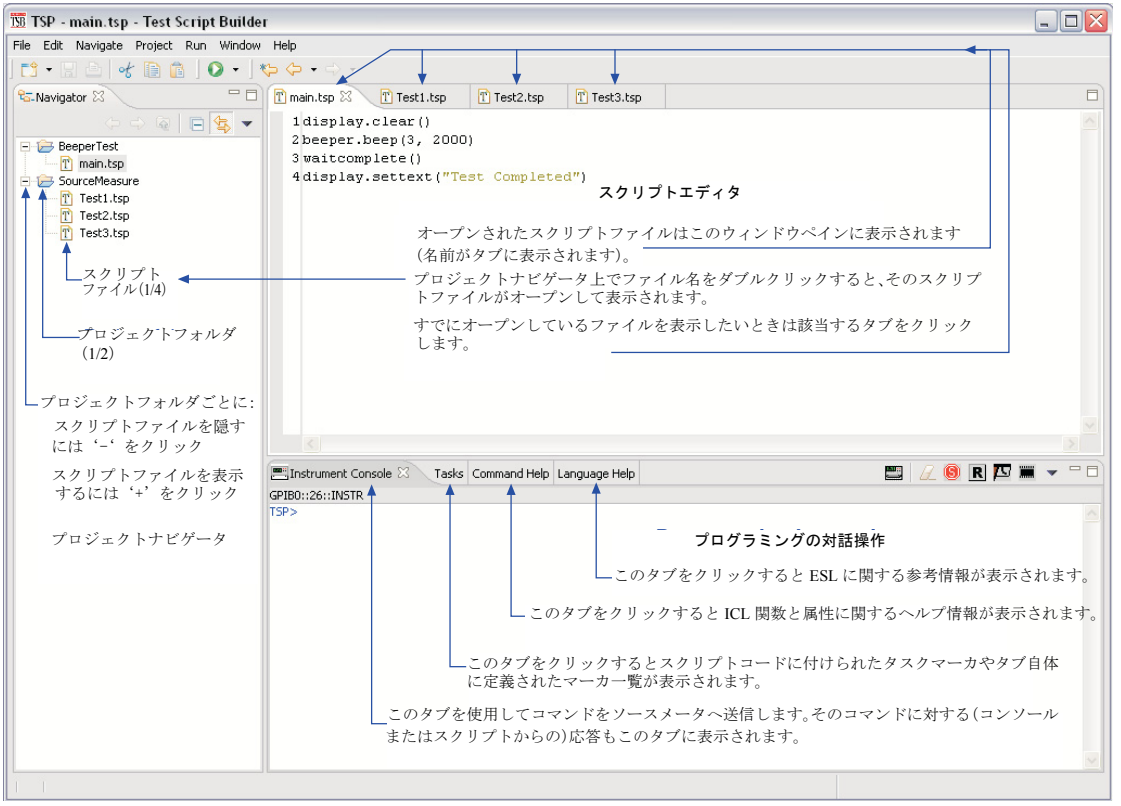
スクリプト エディタ

スクリプトチャンクの作成 / 変更に使用するのがスクリプトエディタです。オープンされたそれぞれのスクリプトファイルに対応するタブが表示されることに注意してください。スクリプトプロジェクトをソースメータへダウンロードすれば実行可能となります。

装置との対話プログラミング

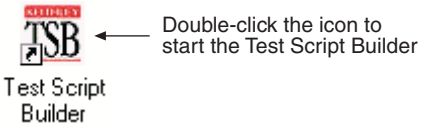
ワークスペースの下側ウィンドウペインには 7 つまでのタブが表示され、Test Script Builder とソースメータ間でそれぞれのタブに対応したやり取りをしながらプログラムを作成してゆくことができます。Instrument Console（図 2-5 で開いているタブ）は接続されているソースメータにコマンドを送信し、コマンドとスクリプトからの受信データ（読み取り値）もこのコンソールに表示されます。それ以外のタブの使用法について詳しくは「対話処理プログラミング タブ」（p.2-30）をご覧ください。

図 2-5
Test Script Builder 画面例



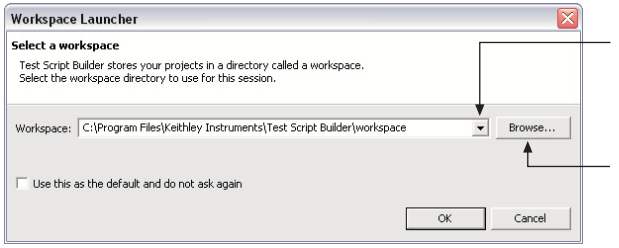
Test Script Builder の起動

ソースメータが正しく PC に接続され、電源が入っていることを確認します（「システム接続」、p.2-10、参照）。PC のデスクトップ上に表示された Test Script Builder アイコンをダブルクリックしてソフトウェアを起動します。



注 **Windows タスクバーのスタートボタンを使用して Test Script Builder を起動することも可能です。デフォルト設定を使用してインストールした場合は以下の順にメニューを操作して Test Script Builder を起動します。**
Start > Program > Keithley Instruments > Test Script Builder

ワークスペース ランチャ –TSB の起動動作中、画面にワークスペース ランチャ（Workspace Launcher）ウィンドウ（下図）が表示され、この中にワークスペースへのパスが表示されます。ここに示されたパスがプロジェクトとスクリプトファイルの保存先になります。以後の起動操作中にこのウィンドウを見る必要がないのであれば、“Use this as the default and do not ask again” にチェックマークを付けてください。OK をクリックして次の起動操作を継続します。



今までに使用されたワークスペースを表示させるにはこのボタンをクリックしてください。使用履歴の新しい方から 5 つのワークスペースが一覧表示されます。

ファイルシステム中の任意のワークスペースを検索するには “Browse…”ボタンを押します。

注：ワークスペースを追加作成する方法については「ワークスペースの新規作成」（p.2-34）をご覧ください。

通信 – Test Script Builder が最初に開いた状態ではソースメータとの通信が閉じられており、このままではコマンドをソースメータへ送ることができません。このままでも Test Script Builder を使用してスクリプトを書くことはできますが、実行することはできません。ソースメータとの通信を確立する方法については「通信回線を開く」をご覧ください。

通信回線を開く

Test Script Builder とソースメータ間の通信をアクティブにするためには装置をオープンする必要があります。通信回線のオープン / クローズには Instrument Console タブのツールバーを使用します。

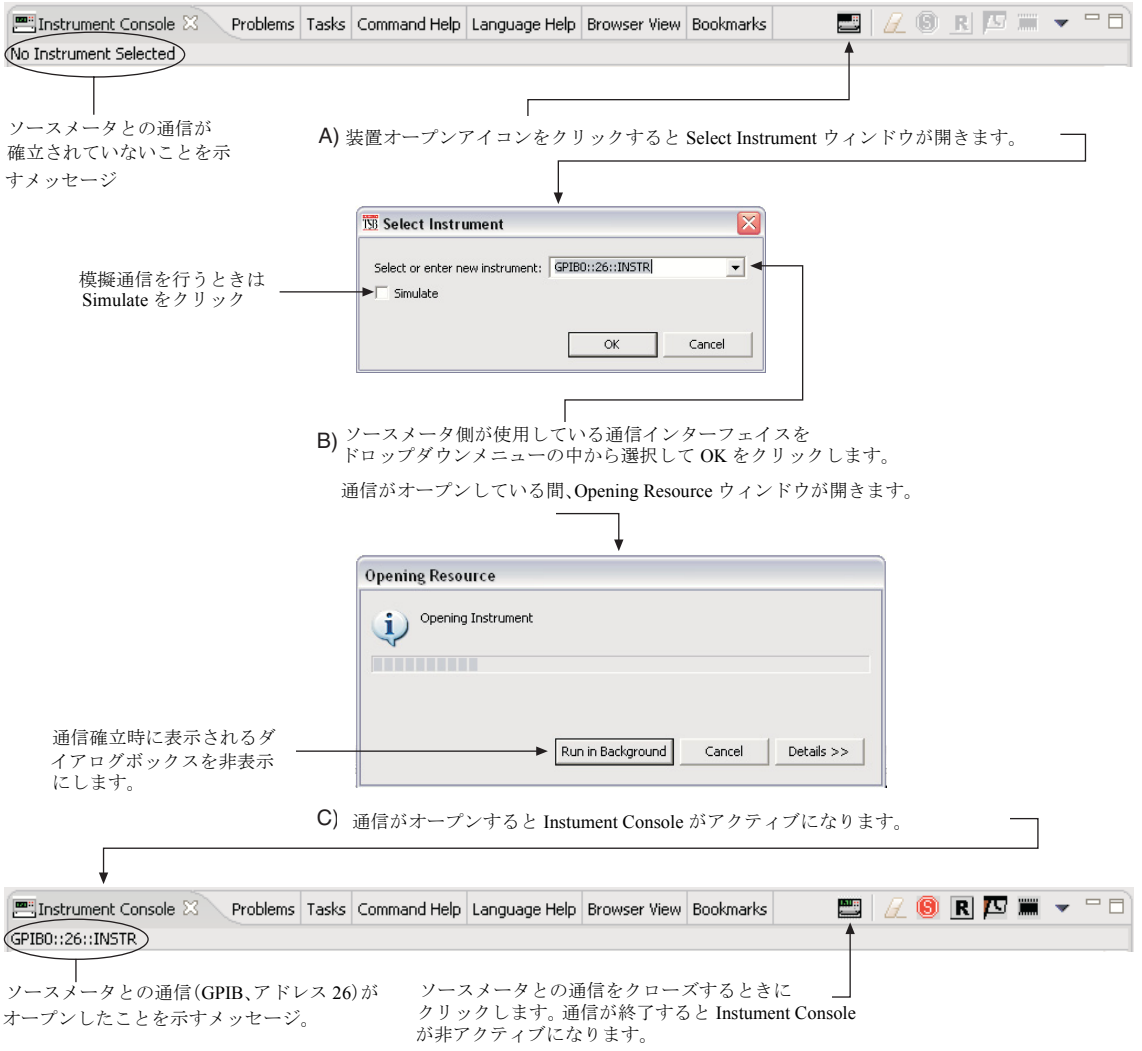
通信をオープン / クローズする具体的な手順が図 2-6 に示されています。この図を補足する詳細情報を以下に説明します。

Select Instrument ウィンドウのドロップダウン メニューを使用して 260x 型で設定されているインターフェイス（GPIB/RS-232）を選択します。

通信シミュレーション - Select Instrument ウィンドウの Simulate オプションを選択すると、ソースメータとの通信が実際には行われなくても Instrument Console がアクティブになります。これによりスクリプトやコマンド送信を模擬的に実行することが可能となりますが、ソースメータは応答しません。

注 TSB とソースメータ間の通信のオープン / クローズに Menu アイコンのドロップダウン メニューを使用することもできます。Menu アイコンの使用法について詳しくは「インスツルメンツコンソール（Instrument Console）アイコン」（p.2-28）をご覧ください。

図 2-6
通信のオープン / クローズ



スクリプトの作成と変更

Test Script Builder を使用してスクリプトの作成 / 変更を行う基本的な手順を図 2-7 のフローチャートに示します。チャート中のラベル（A-G）は参照項目との対応を示します。

図 2-7
Test Script Builder を使用するスクリプトの作成 / 変更

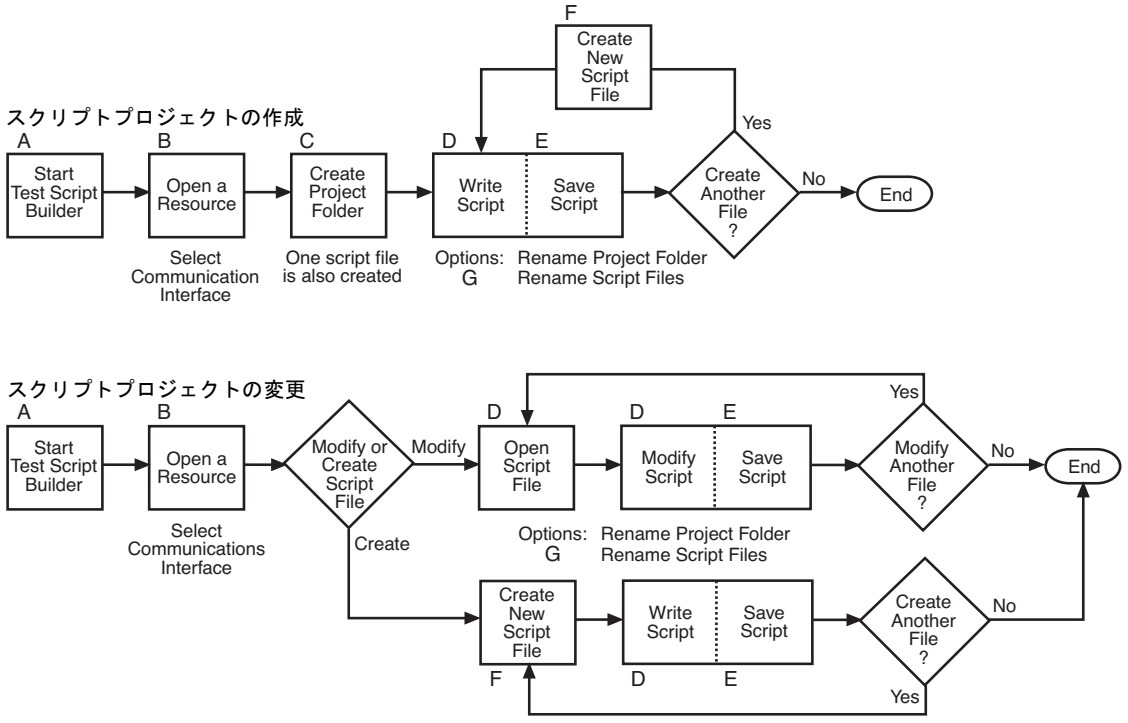


図 2-7 のラベル A から G に対応する参照項目：

A 「Test Script Builder の起動」- 詳細：p.2-14
B 「通信回線を開く」- 詳細：p.2-15
C 「プロジェクトフォルダの作成」- 詳細：p.2-18
D 「スクリプトの作成 / 変更」- 詳細：p.2-19
E 「スクリプトの保存」- 詳細：p.2-19
F 「スクリプトファイルの新規作成」- 詳細：p.2-20
G 「プロジェクトフォルダ / スクリプトファイルの名前の付け替え」- 詳細：p.2-21

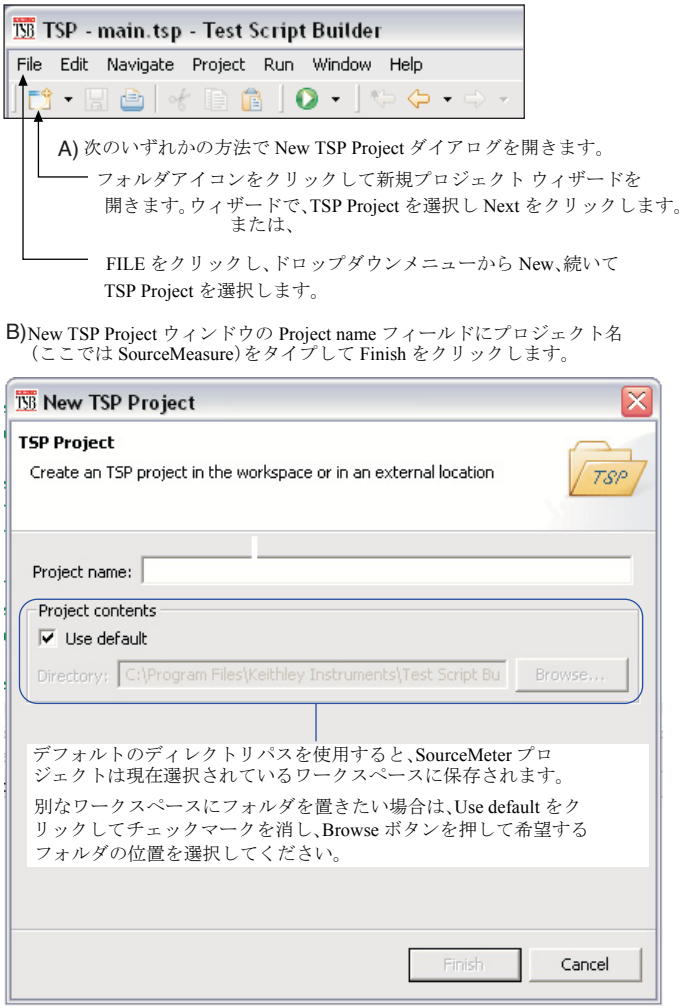
プロジェクトフォルダの作成

プロジェクトフォルダを作成すると同時に以下の処理が行われます。

- プロジェクトフォルダがプロジェクトナビゲータに追加されます。
- プロジェクトフォルダ内にスクリプトファイル（名前：“main”）が作成されます。
- 空のスクリプトファイルがオープンして Test Script Builder の Script Development ペインに表示されます。

プロジェクトフォルダの作成には Test Script Builder の一番上に配置されたツールバーを使用します。具体的なプロジェクトフォルダ作成法を図 2-8 に示します。

図 2-8
プロジェクトフォルダの作成



スクリプトの作成 / 変更

スクリプトは ICL コマンドと TSL 文によって構成されます。図 2-5 に示すのは簡単なスクリプトの例であり、これを実行するとブザーのテストが行われます。1 KHz のブザーを 3 秒間鳴動させてから “Test Completed” というメッセージを 260x 型に表示します。更に詳しくは「ユーザスクリプト」（p.2-39）をご覧ください。

プロジェクトまたはスクリプトファイルを作成すると、スクリプトファイルがオープンして Test Script Builder の Script Development ペインに表示されますから、ここにスクリプトを記述します。

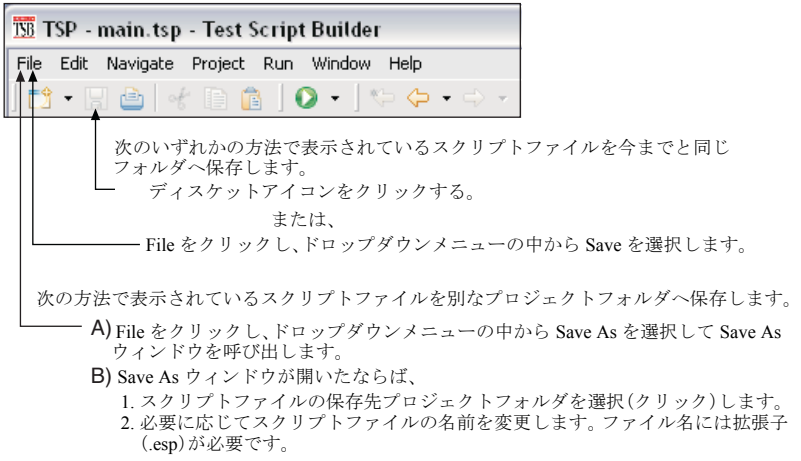
既存のスクリプトファイルを変更したい場合は、まずそのファイルをオープンする必要があります。Script Editor のタブには現在オープンしているスクリプトファイル名が表示されています。新たにスクリプトファイルをオープンするには、プロジェクトナビゲータに表示されているファイル名をクリックしてください。すでにオープンしている別なスクリプトファイルを表示するには、スクリプトエディタ上端のタブの中から該当する名前のタブをクリックしてください。

スクリプトの保存

コードの作成 / 変更がある程度進むたびに面倒がらずにスクリプトを保存するようにお奨めします。保存するたびにスクリプトのチェックが行われ、エラーが発見されると問題のある行の近くに “X” が表示されるとともに、Problems タブが開いて発生したエラーの内容を説明してくれます。“X” はプロジェクト ナビゲータにも表示され、どのプロジェクトフォルダのどのスクリプトに問題があるかが一目で分るようになっています。

現在表示されているスクリプトファイルを保存するには Test Script Builder 上端に配置されたツールバーを使用します。図 2-9 で説明したように、スクリプトは同じフォルダに保存するばかりでなく、別なフォルダを選択して保存することも可能です。

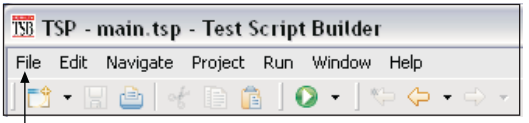
図 2-9
Test Script Builder にスクリプトを保存する



スクリプトファイルの新規作成

スクリプトプロジェクトは複数のスクリプトファイルを含むことがあります。プロジェクトフォルダにスクリプトファイルを新しく追加する方法を図 2-10 に示します。

図 2-10
スクリプトファイルの新規作成



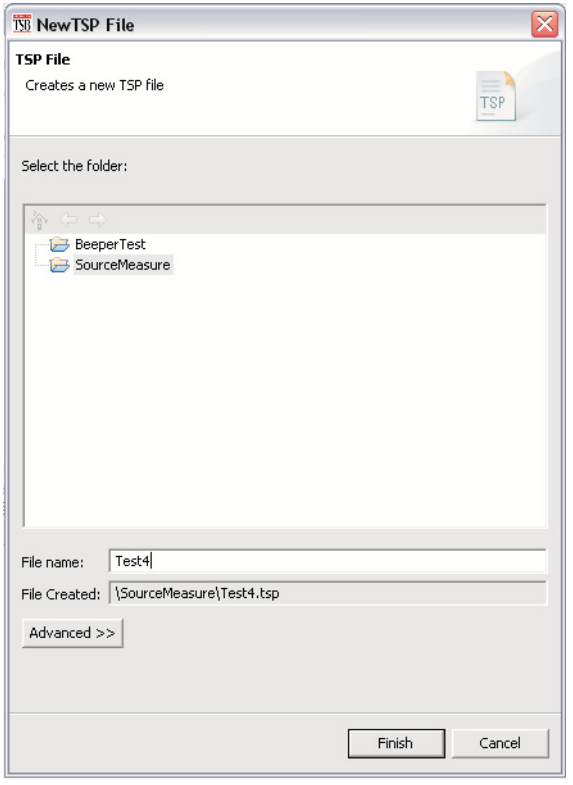
A) 次のいずれかの方法で New TSP File ウィンドウを開きます。
FILE をクリックし、表示されたドロップダウンメニューの中から New、続いて TSP File を選択します。

または、

スクリプトファイルを追加する対象となるフォルダをプロジェクトナビゲータ内で右クリックし、表示されるドロップダウンメニューの中から New、続いて TSP File をクリックします。

B) New TSP File ウィンドウが開いたならば、希望するプロジェクトフォルダが選択されていることを確認してください。フォルダを選択するにはそれをクリックします。

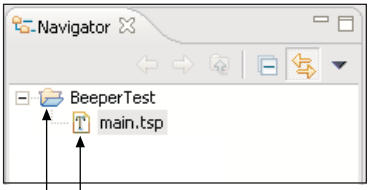
C) ファイル名（ここでは Test4）をタイプ入力して Finish をクリックします。



プロジェクトフォルダ / スクリプトファイルの名前の付け替え

プロジェクトを新たに作成すると、“main”という名前でスクリプトが自動的に作成されて、そのフォルダに保存されます。図 2-11 に示すのは、1 つのプロジェクトフォルダと、新たに作成されてプロジェクトナビゲータに追加されたスクリプトファイルです。ここに示すように、プロジェクトフォルダとスクリプトファイルは名前を変更することができます。

図 2-11
プロジェクトフォルダ / スクリプトファイルの名前の付け替え



スクリプトファイルの名前変更:
A) スクリプトファイルを右クリックし、表示されるドロップダウンメニューの Rename を選択します。
B) 新しい名前をタイプ入力(拡張子 .tsp を忘れないでください)してから Enter キーを押します。

プロジェクトフォルダの名前変更:
A)プロジェクトフォルダを右クリックし、表示されるドロップダウンメニューの Rename を選択します。
B)新しい名前をタイプ入力してから Enter キーを押します。

スクリプト起動条件の設定

スクリプト実行のためには、まず 260x 型へのローディングが必要となります。スクリプト起動時の設定のために以下のオプションが用意されています。

- 起動時に含めるスクリプトファイルの選択
- 選択したスクリプトファイルの起動順序設定
- スクリプト起動時にローディングのみを行うか、実行も含めるかの設定
- スクリプトを 260x 型の揮発性メモリに保存するか、不揮発性メモリに保存するかの設定。揮発性メモリに保存されたスクリプトはソースメータの電源が切れると同時に失われますが、不揮発性メモリに保存しておけば電源が切れても失われることはありません。

スクリプトプロジェクトを作成した時点での起動時設定は、次のようになっています。

- 最初のスクリプトファイル（“main”）のみが選択されて起動パッケージに含まれています。
- ローディング後、即実行されます。
- スクリプトプロジェクトは 260x 型の揮発性メモリに保存されます。したがって、260x 型の電源が切れると同時にスクリプトも失われます。

注　　初期起動条件設定がユーザの要求にマッチしているのであれば、「スクリプトの起動」（p.2-25）に説明する方法で直ちにスクリプトを起動することができます。

図 2-12 のフローチャートは、スクリプトの起動条件設定を変更するための基本的な操作手順を示しています。チャート中のラベル（A-G）は下の説明文との対応を示します。

図 2-12
起動条件設定の変更

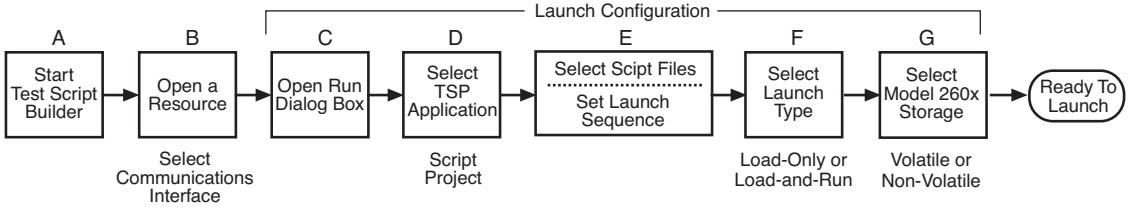


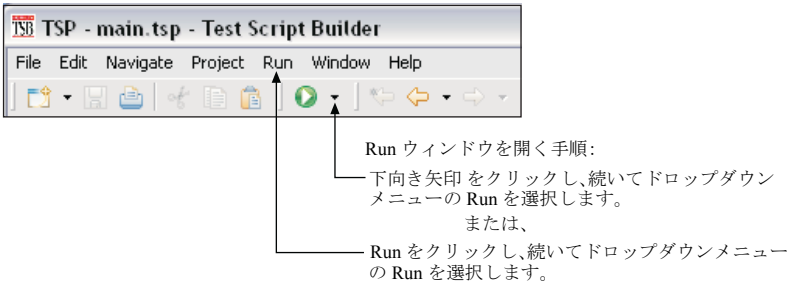
図 2-12 のラベル A から G に対応する参照項目：

A 「Test Script Builder の起動」– 詳細：p.2-14
B 「通信回線を開く」– 詳細：p.2-15
C 「起動条件設定ウィンドウの表示」– 詳細：p.2-23
D 「設定の選択」– 詳細：p.2-24
E 「スクリプトファイルの選択と起動順序」– 詳細：p.2-24
F 「起動タイプの選択」– 詳細：p.2-24
G 「スクリプトの保存」– 詳細：p.2-24

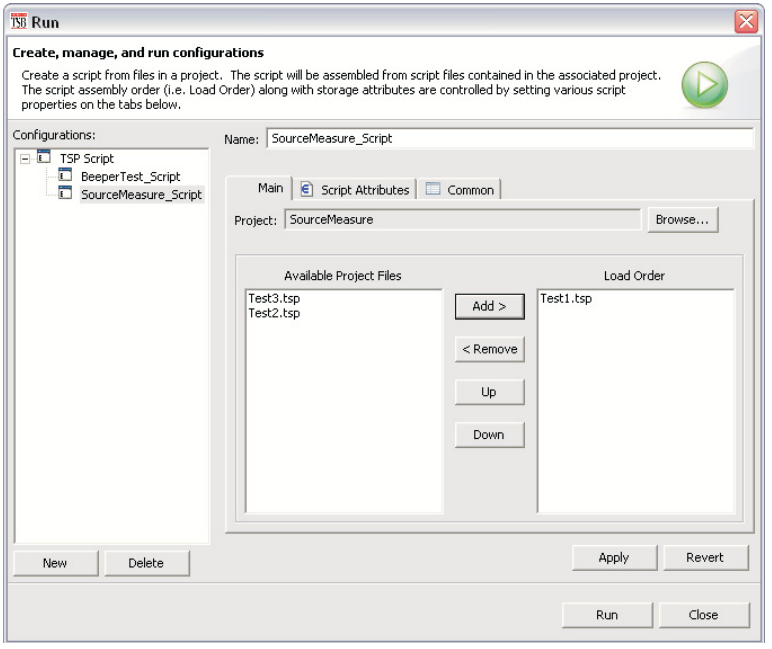
起動条件設定ウィンドウの表示

起動条件の設定には Run ダイアログボックスを使用します。図 2-13 に示すように、起動条件を設定するウィンドウを開くには Test Script Builder 画面の上端に配置されたツールバーを使用します。

図 2-13
Run ダイアログボックスを開く（起動条件設定）



起動条件の設定 ー Main タブを表示した状態



設定の選択

Test Script Builder を使用してプロジェクトを作成すると、同時にその起動条件を定義する設定（Configuration）ファイルも作られます。この設定ファイルにはプロジェクト名に “_Script”を付加した名前が付けられます。例えば、プロジェクト名が“SourceMeasure”であれば設定ファイルの名前は“SourceMeasure_Script”になります。

Run ウィンドウの Configurations ペインに TSP スクリプト一覧が表示されます。あるスクリプトの起動条件を表示させたい場合は、該当する Configurations 名をクリックしてください。図 2-13 に示すのは “SourceMeter_Script”に対応する Main タブです。

スクリプトファイルの選択と起動順序

図 2-13 に示すように、プロジェクトに属するスクリプトファイルは設定ウィンドウの Main タブに表示されます。このタブの Available Project Files 側ペインに表示されるのは、起動時に選択されていないスクリプトファイル一覧です。一方、Load Order 側に表示されたスクリプトファイルは、起動時に選択されているスクリプトであり、表示順に実行されます。

Main タブ内の設定を変更したい場合は以下の操作を行います。

- スクリプトファイルを Load Order 側へ移動させるには、対象となるファイル名をクリックしてから “Add >”ボタンを押します。
- ファイルを Available Project Files 側へ移すには、対象となるファイル名をクリックしてから “< Remove”ボタンを押します。
- Load Order 側に登録されたファイルの起動順序を変更したい場合は、同じようにファイルを指定してから Up/Down ボタンを押します。
- Main タブ内で行った変更は Apply ボタンを押すことによって確定します。

起動タイプの選択

起動プロセスについては 2 通りのオプションが用意されています。

- Load – スクリプトは 260x 型のランタイム環境にローディングされますが、即実行はされません。スクリプトは後で実行されます。
- Load and Execute – スクリプトはランタイム環境にローディングされ、ローディングプロセス完了後直ちに実行されます。
- Auto Run – Load and Execute を選択すると Auto Run オプションが選択可能になります。この項目を選択しておくと、260x 型が起動される度にそのスクリプトが自動実行されます。

スクリプトの保存

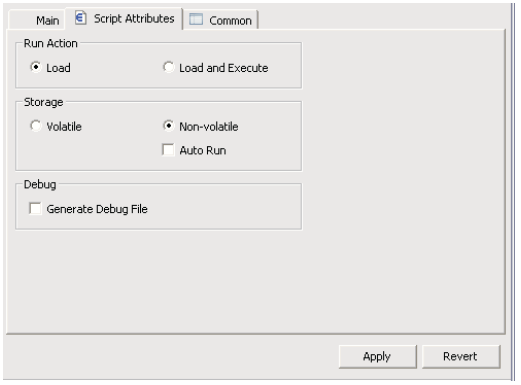
起動されたスクリプトは、260x 型の揮発性または不揮発性メモリのいずれかに保存されます。揮発性メモリに保存されたスクリプトは、ソースメータの電源が切れると同時に失われますが、不揮発性メモリに保存しておけば電源が切れても失われることはありません。

スクリプトの保存法は Run ウィンドウの Script Attributes タブから設定します（図 2-14 参照）。このタブの中にある Volatile（揮発性）または Non-volatile（不揮発性）を選択してください。非揮発性メモリを選択すると Auto Run オプションが選択可能（チェック可能）になります。このオプションを選択するとソースメータ起動時にスクリプトが自動実行されます。

Debug - Generate Debug File オプションをチェックするとスクリプトの読み出し専用コピーが作成されるとともに “Debug” という名前のフォルダとデバッグファイル（.DBG）がプロジェクトに追加されます。

保存設定を変更したときは、必ず Apply をクリックして変更内容を確定させてください。

図 2-14
Run ダイアログボックス（Script Attributes タブ）

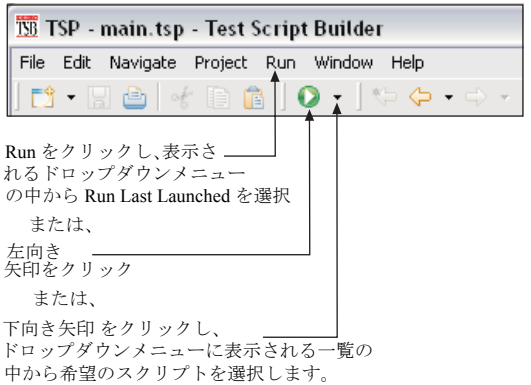


スクリプトの起動

起動条件設定のチェック / 変更が終われば、Run ダイアログボックスの Run ボタン（図 2-13）をクリックしてスクリプトを起動することができます。

別法として、Test Script Builder 画面上端のツールバーを使用して直接再起動することも可能です。ツールバーからスクリプトを再起動する手順を図 2-15 に示します。

図 2-15
Test Script Builder ツールバーを使用してスクリプトを再起動



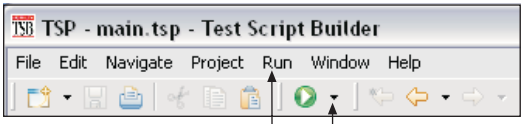
TSP ファイルの実行

TSP（.tsp）ファイルを実行するには必ずしも全体を 260x 型装置にローンチ（ローティング）する必要はなく、TSP ファイルをコード単位で 260x 型に送ることで実行が可能です。この場合、TSP ファイルは 260x 型上に常駐（揮発性または不揮発性メモリに保存）しません。TSP ファイルを実行するにはプロジェクトナビゲータまたは Test Script Builder 画面上端のツールバーを使用します。

プロジェクトナビゲータから TSP ファイルを実行するときは、まず .tsp ファイル（main.tsp）を右クリックし、次にマウスメニューの Run を選択してから最後にサブメニューの Run As TSP File をクリックします。

TSB ツールバーから TSP ファイルを実行することができます（図 2-16 の説明参照）。

図 2-16
Test Script Builder ツールバーを使用してスクリプトを再起動



Run または下向き矢印をクリックしてドロップダウンメニューを表示させ、Run As を選択してからサブメニューの 1 TSP File を選択します。

Instrument Console ツールバーの Menu アイコンを使用して TSP ファイルを実行することも可能です。詳細については「インスツルメンツコンソールのアイコン」（p.2-28）の説明をご覧ください。

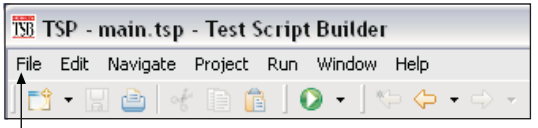
260x 型装置からスクリプトを読み出す

ユーザスクリプトとファクトリスクリプトのどちらも 260x 型のメモリから読み出すことができます。読み出したスクリプトフォルダはプロジェクト ナビゲータに登録され、対応するスクリプトファイルが開きます。

260x 型からスクリプトをインポートする具体的な手順を図 2-17 に示します。ソースメータとの通信はすでにオープンされていなければなりません。通信がクローズ状態でインポート作業を始めると、通信をオープンするためのウィンドウが画面に表示されます。

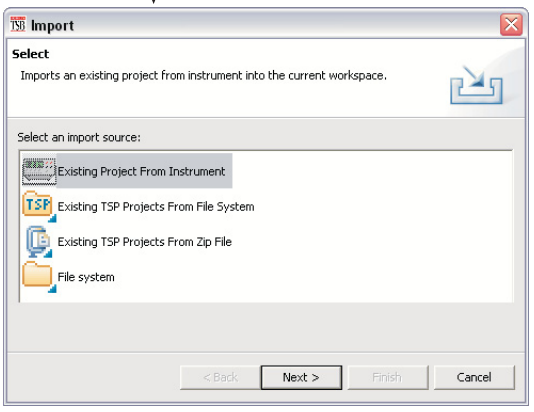
変更後のスクリプトを再び 260x 型へローディングする場合は同名で保存することは勿論、名前を変えてローディングすることも可能です。ファクトリスクリプトをインポートしてそれを 260x 型へ再度ローディングする場合は、ユーザスクリプトとしてのみ保存が可能です。

図 2-17
260x 型からスクリプト（ここでは KIGeneral_Script）をインポートする方法

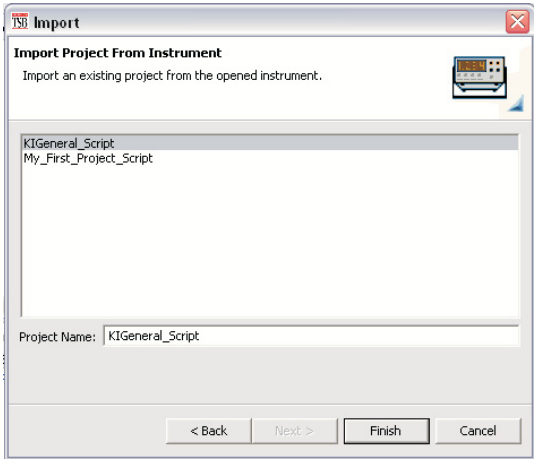


A) File をクリックしてドロップダウンメニューを表示させ、Import を選択して Import ウィザードを呼び出します。

B) Import の Select ボックスの中に表示された “Existing Project From Instrument” をクリックし、続いて Next をクリックします。



C) Import Project From Instrument ボックスで KIGeneral_Script を選択してから Finish をクリックします。



インスツルメンツコンソール（Instrument Console）

ソースメータとの通信が確立していれば、Instrument Console を使用して以下の操作を行うことができます。

- チャンクの実行：この場合のチャンクは個別の ICL コマンドと TSL プログラミング文を意味します。
- 返送されたデータ（読み取り値、メッセージ）の表示。
- Instrument Console から誤りを含むコードを送信したことにより発生したエラーメッセージの表示。

インスツルメンツコンソールを開くには Test Script Builder ウィンドウの下側ペインにある Instrument Console タブをクリックします（図 2-5）。

Instrument Console がアクティブになると TSP> プロンプトが表示されます。プロンプトに続いてコマンドをタイプ入力してから、Enter を押すことにより実行されます。一例として次のコマンドを入力しください。

```
TSP>reset ()
```

Enter を押すとソースメータがリセットされて、デフォルト設定の状態に復帰します。

注　　Instrument Console から送信できるコード例が「コマンドと文の送信」の項に説明されています。

Instrument Console に書き込んだコードやメッセージを消すには図 2-18 に示すコンソールウィンドウ消去（Clear Console Window）アイコンをクリックしてください。マウスメニューを以下のように操作して消すことも可能です。マウスポインタをコンソールウィンドウ上に移動させて右クリックし、表示されるマウスメニューの中から Clear Console Window を選択します。

Instrument Console アイコン

ソースメータとの通信がオープンされると、Instrument Console ツールバー（図 2-18）のすべてのアイコンがアクティブになります。

図 2-18
Instrument Console アイコン



Instrument Console に現われるアイコンの説明：

装置クローズ – 通信がオープンしている状態でこのアイコンをクリックすると、ソースメータとの通信がクローズされて通信できなくなります。

コンソールウィンドウをクリア – このアイコンをクリックするとすべてのコードと Instrument Console ウィンドウからの応答メッセージがすべて画面から消去されます。別な 2 つの方法でも同様に Instrument Console ウィンドウをクリアすることができます。

- カーソルをコンソールウィンドウ上へ移動させて右クリックし、表示されるマウスメニューから Clear Console Window を選択します。
- 図 2-18 のメニューアイコンをクリックし、表示されるメニューの中から Clear Console Window を選択します。

実行中断 – このアイコンをクリックすると、Instrument Console から送信したコマンドの実行が中止されます。

リセット – このアイコンをクリックすると、ソースメータがリセットされます。reset() コマンドを送信するのと同じ機能です。

ソフトウェアトリガ送信 – このアイコンをクリックすると、ソースメータへソフトウェアトリガが送信されます。このマニュアルのセクション 4「トリガ」をご覧ください。

NVRAM からスクリプトを消去 – このアイコンをクリックすると、ソースメータの不揮発性メモリからスクリプトが消去されます。アイコンをクリックするとスクリプト一覧が表示されますから、削除したいスクリプトを選択して Delete をクリックしてください。

メニュー – このアイコンをクリックすると、以下の項目を持つメニューが表示されます。

- Clear Console Window – コンソールウィンドウの内容を消去します。コンソールをクリアする別な方法が上記の「コンソールウィンドウをクリア」アイコンの項に説明されています。
- Instrument – このメニューをクリックすると、サブメニューが表示され、他のツールバーアイコンを使用するのと同じ操作を実行することができます。メニューに含まれる Flash を選択して Keithley Flash Programmer を使用すれば、ファームウェアアップグレードを 260x 型へダウンロードすることができます。その使用法について詳しくは「Flash Programmer」（p.2-34）をご覧ください。
- Save Console – Instrument Console ウィンドウの内容（コードと応答メッセージ）をテキストファイル（.txt）として保存します。このメニューをクリックすると保存先選択ダイアログが表示されますから、保存先を指定してログを保存します。保存されたテキストファイルは任意のテキストエディタ（WordPad など）で表示することができます。
- Run – プロジェクトナビゲータやコンピュータ内の他の場所（もしくはネットワーク上）に存在する任意の TSP（.tsp）ファイルを実行したいときにこのメニューを使用します。Run をクリックすると Editor または Script File を選択するサブメニューが表示されます。プロジェクトナビゲータに含まれるプロジェクトに関する項目もこのサブメニューに表示されます。
  - Editor – この項目を選択すると別なサブメニューが開き、プロジェクトナビゲータ内に存在する全 TSP ファイルの一覧が表示されます。スクリプトファイルをクリックすると、そのスクリプトが実行されます。

- Script File – この項目を選択するとファイル選択ダイアログが開きますから、コンピュータ内またはネットワーク上に保存されている TSP ファイルを指定してください。ダイアログの File Name 欄に希望の TSP ファイル名が表示された状態で Open をクリックすると、その TSP ファイルが実行されます。
- Projects – Run メニューをクリックするとプロジェクトナビゲータ内に登録されているプロジェクト一覧が表示されます。プロジェクトを選択すると、そのプロジェクトに属する TSP ファイルが表示されますから、TSP ファイル名をクリックすればそのファイルが実行されます。

Problems、Tasks、Bookmarks タブのいずれかを開いた（表示した）ときにもメニューアイコンが表示されます。

対話処理プログラミングタブ

ワークスペースの下側ウィンドウペインには 7 つまでのタブが表示され、Test Script Builder とソースメータ間でそれぞれのタブに対応した対話操作を行いながらプログラムを作成してゆくことができます。

ワークスペース上に配置できるタブには例えば次の項目が含まれます。Instrument Console、Problems、Tasks、Command Help、Language Help、Browser View、および Bookmarks。現在ワークスペース上に現われていないタブを追加したいときは、ワークスペース上端のツールバーの Window オプションからスタートして以下の操作を行います。

Window をクリック > Show View を選択 > 表示したいタブをクリック

ワークスペース上のタブを開く（表示する）にはそのタブ名をクリックしてください。あるタブが開くと、そのタブ名の右側に “X” シンボルが表示されます。“X” をクリックすると、そのタブがワークスペースから削除されます。

Instrument Console タブ

このタブ（図 2-5）を使用して接続されているソースメータへコマンドを送信します。コマンドとスクリプトの実行により返されたデータ（読み取り値）も Instrument Console に表示されます。

注 以下のタブの画面表示の一部が図 2-19 および図 2-20 に示されています。

Problems タブ

スクリプトファイルを保存するときにエラーチェックが行われます。ここでエラーが検出されると、スクリプトエディタに表示された問題行 / コード（またはそのすぐ近く）の左端に “X” が表示されるとともに、Problems タブが自動的に開いてエラーに関する情報を表示します。

Problems タブで問題個所をクリックすると、スクリプトエディタ上の “X” マーカの付いたコード行がハイライト表示されます。問題のある行を修正してスクリプトを保存することにより、問題点の表示も画面から消去されます。

Tasks タブ

このタブには特定のファイルやその特定の行に付随するユーザ定義タスク、または特定のファイルに付随するのではない汎用タスクが表示されます。

希望するコード行に対応する行番号を右クリックし、マウスメニューの Add Task を選択することにより、選択した行に対応するスクリプトエディタの左端にタスクマーカ（√）が挿入されます。New Task ウィンドウが表示されますから、タスクの説明文をタイプ入力してから OK をクリックしてください。以上の操作によってタスクが Task タブに追加されます。Tasks タブであるタスクをクリックすると、スクリプトエディタ上で該当するタスクマーカの付いたコード行がハイライト表示されます。あるタスクをスクリプトエディタから削除したい場合は、タスクマーカを右クリックしてから Remove Task を選択してください。

特定のコードやファイルにリンクされていないタスクを Tasks タブに追加することも可能です。その場合にはマウスカーソルを Tasks タブへ移動して右クリックし、Add Task を選択してタスクの説明を入力します。

Command Help タブ

このタブは ICL 関数と属性の詳しい情報を提供します。このヘルプの内容はリファレンスマニュアルのセクション 12 と同じものです。Command Help の最初のページにはヘルプファイルの主要トピックスへのリンク一覧が表示されます。ICL コマンドのリストをクリックして関数と属性一覧を表示させ、さらにいずれかの関数または属性をクリックして詳細情報を表示させます。

Language Help タブ

このタブは TSL（Test Script Language）の詳細情報を提供します。含まれる内容はこのセクションの「TSL（Test Script Language）リファレンス」（p.2-52）の説明と同じものです。Language Help の最初のページにはヘルプファイルの主要トピックスへのリンク一覧が表示されます。

Browser View タブ

インターネット接続がオンラインになっているときにこのタブを使用すれば簡単に Keithley の Web サイト（www.keithley.com）を閲覧することができます。

Bookmarks タブ

Tasks タブ

ユーザがスクリプトエディタ上に設定したブックマークを表示したいときにこのタブを使用します。希望するコード行に対応する行番号を右クリックし、マウスメニューの Add Bookmark を選択することにより選択した行に対応するスクリプトエディタの左端にブックマークが付けられます。Add Bookmark ウィンドウが現われますから、ブックマークに名前を付けてから OK をクリックしてください。以上の操作により、ブックマーク名が Bookmarks タブに追加されます。

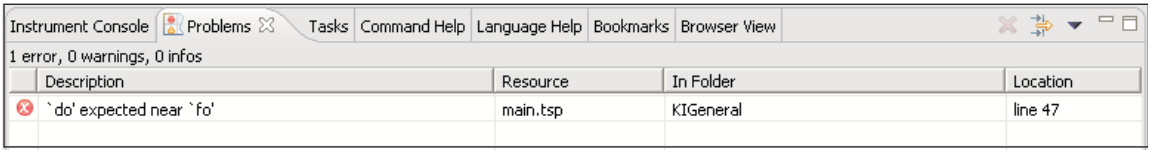
Bookmarks タブ上のブックマークをクリックすると、そのブックマークが付いたコード行がハイライト表示されます。あるブックマークをスクリプトエディタから削除したい場合は、該当するブックマークを右クリックしてから Remove Bookmark を選択してください。

図 2-20 の Bookmarks タブはブックマークの使用例を示しています。タブ内のそれぞれのブックマークはプロジェクトナビゲータ内に存在するスクリプトファイ

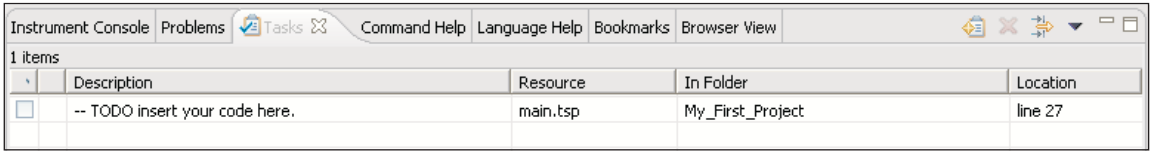
ルの関数にリンクされています。あるブックマークをクリックすると、対応する関数の先頭行がスクリプトエディタ上にハイライト表示されます。

図 2-19
プログラミング操作に関するタブ：Problems、Tasks、Command Help

Problems tab:



Tasks tab:



Command Help tab:

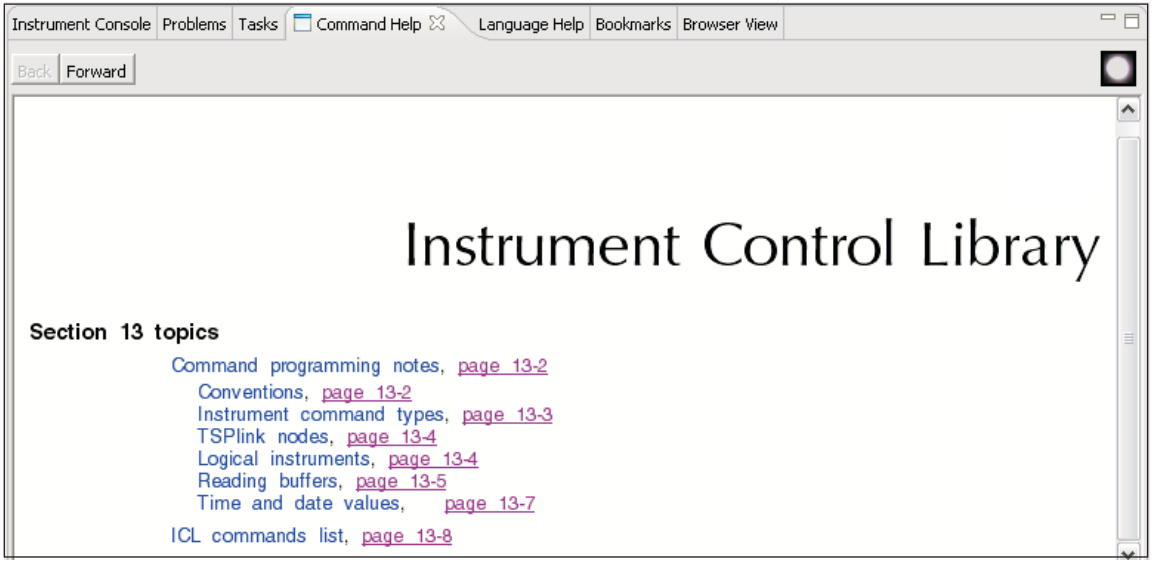
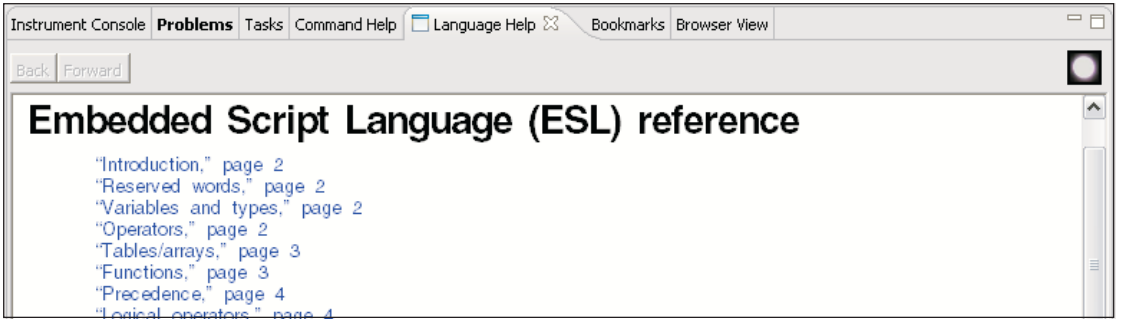
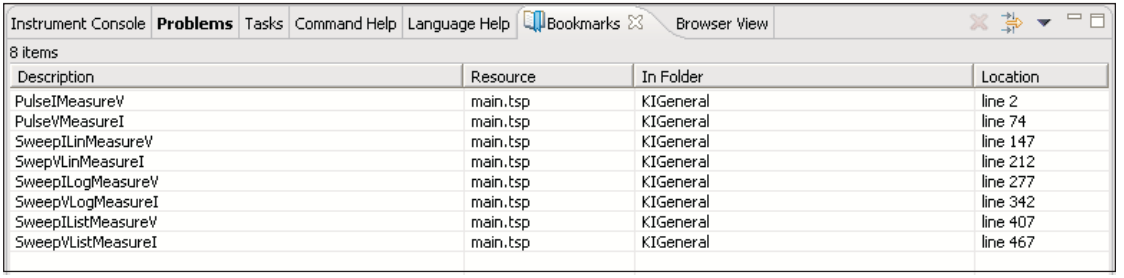


図 2-20
プログラミング操作に関するタブ：Language Help、Bookmarks、Browser View

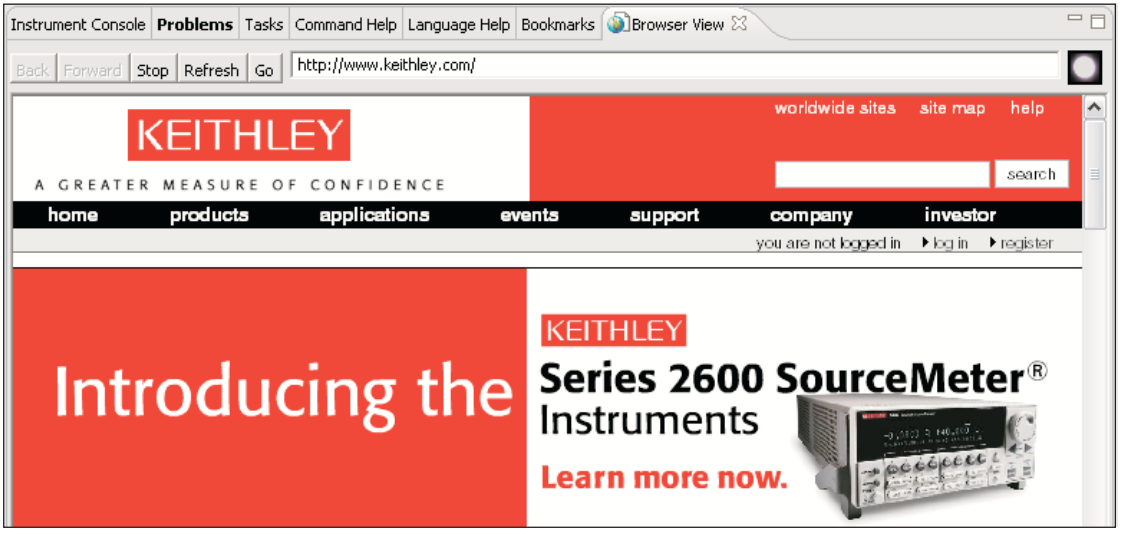
Language Help tab:



Bookmarks tab:



Browser View tab:



Flash programmer

260x 型のファームウェアがアップグレードされたときは、その最新版を Keithley の Web サイト（www.keithley.com）からダウンロードして頂けます。アップグレードには新規作成または改良版スクリプトが含まれることもあります。ダウンロードされたファームウェアアップグレード ファイルは flash programmer を使用して 260x 型へインストールされます。

注意　外部回路が入出力端子に接続された状態でフラッシュアップグレードを行うと、装置や DUT に損傷を与える可能性があります。フラッシュアップグレード実行の前に必ず入出力端子の接続を取り外してください。

TSB- ソースメータ間の通信がオープンされていれば、メニューアイコン（図 2-18）を使用して flash programmer へ以下のようにアクセスすることができます。

メニューアイコンをクリック > Instrument を選択 > Flash をクリック

表示されるファイル選択ダイアログを使用して、ダウンロードされたファイルを指定してから Open をクリックしてアップグレードを開始します。更に詳しくは「フラッシュファームウェア アップデート」（p.13-13）をご覧ください。

ファイル管理タスク

プロジェクトとそれに付随するファイル（スクリプトファイル）はワークスペースフォルダに存在します。典型的なファイル管理タスクには例えばプロジェクトやスクリプトファイルの新規作成が含まれます。（ファイル管理タスクについてより詳しくは「スクリプトの作成と変更」（p.2-17）をご覧ください。）260x 型からスクリプトプロジェクトを Test Script Builder へインポートして、そこで変更するという方法も可能です（詳細は「260x 型装置からスクリプトを読み出す」（p.2-26）を参照）。

それ以外のファイル管理タスクには「ワークスペースの新規作成」、「別なワークスペースからプロジェクトをインポート」、「ワークスペースの切換え」、「プロジェクト、スクリプトファイルの削除」などが含まれます。これらのファイル管理タスクを以下順に説明します。

ワークスペースの新規作成

ワークスペースはファイルシステムの任意の場所に作成することができます。新しいワークスペースとは単にプロジェクトファイルを保存するための新しいフォルダであり、TSB から次の方法で作成します。

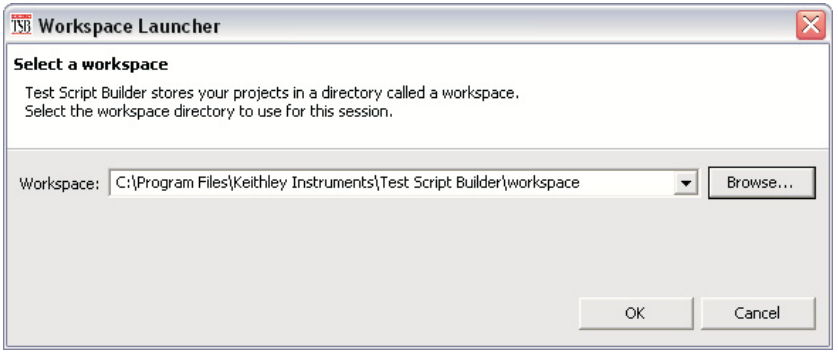
1. TSB の上端に配置されたツールバーの File をクリックしてファイルメニューを開き、Switch Workspace を選択してワークスペースランチャ（Workspace Launcher、p.2-21A）を開きます。
2. Browse ボタンをクリックして Select Workspace Directory ブラウザを開き、フォルダを新規作成する場所を指定します。図 2-21B は新しいワークスペースフォルダの場所として選択された Test Script Builder フォルダを示しています。ワークスペースフォルダはファイルシステムのどこに置いても良いことを憶えておいてください。
3. Select Workspace Directory の中にある Make New Folder ボタンをクリックします。選択されている場所に New Folder という名前でフォルダが挿入されます。
4. このブラウザ内で New Folder を右クリックし、表示されるマウスメニューの Rename を選択します。

5. 新しいワークスペースフォルダの名前（例えば workspace2）をタイプ入力してから Enter を押します。
6. ブラウザ内の OK ボタンをクリックし、続いてワークスペースランチャの OK をクリックします。Test Script Builder が一旦閉じますから、再度新しいワークスペースを使用して起動してください。

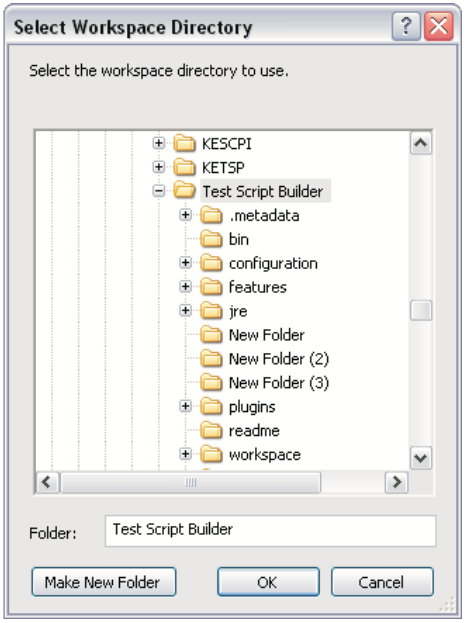
新しいワークスペースのプロジェクトナビゲータにはまだ何もプロジェクトが存在しません。「スクリプトの作成と変更」（p.2-17）で説明した方法で新しいプロジェクトとスクリプトを作ることができます。また、プロジェクト（および付随するスクリプトファイル）を別なワークスペースフォルダから新しいワークスペースへインポートすることも可能です。その方法について詳しくは「別なワークスペースからプロジェクトをインポート」（p.2-36）をご覧ください。

図 2-21
ワークスペースランチャとワークスペース ディレクトリ選択ダイアログ

A)



B)

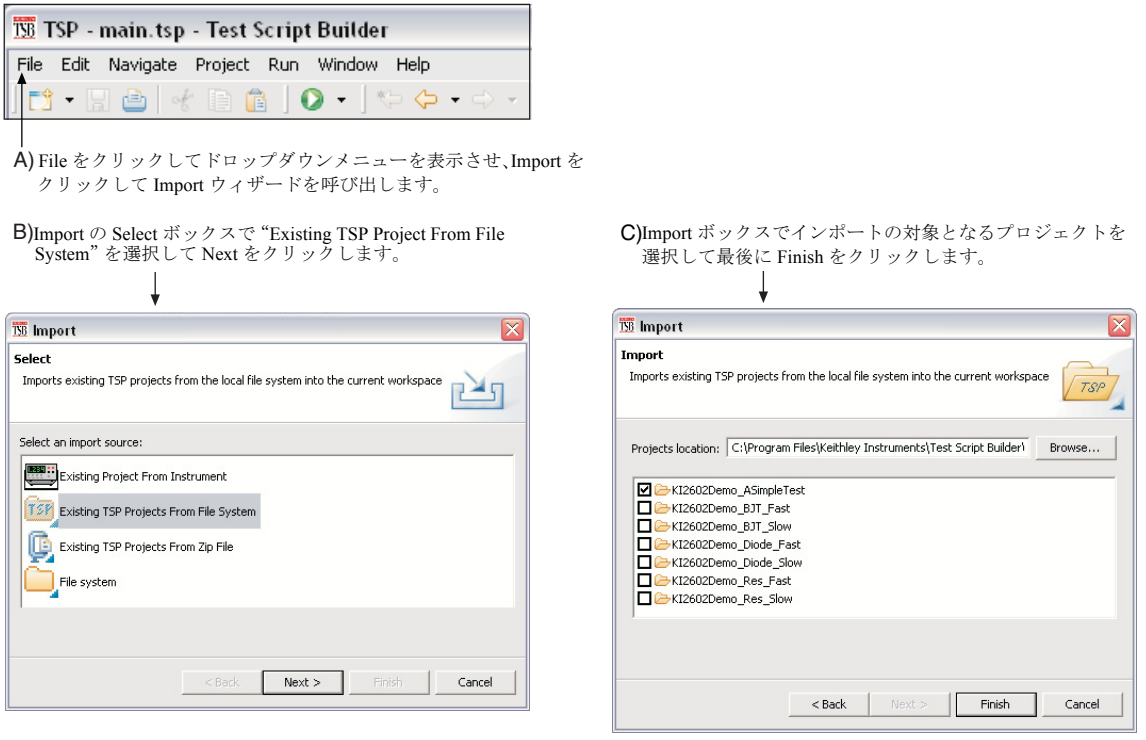


別なワークスペースからプロジェクトをインポート

ファイルシステム内の別なワークスペースフォルダからプロジェクト（および付随するスクリプトファイル）をインポートすることができます。図 2-22 では KI2602Demo_AsimpleTest という名前のプロジェクトを例としてこの手順を説明しています。図のステップ C ではブラウザを使用してインポートしたいプロジェクトの場所を指定しています。

Import ウィンドウの Finish をクリックすることにより、そのプロジェクトが Test Script Builder のプロジェクトナビゲータに現われます。

図 2-22
別なワークスペースフォルダからプロジェクトをインポート



ワークスペースの切換え

別なワークスペースへの切換えには以下の操作を行います。

1. TSB の上端に配置されたツールバーの File をクリックしてファイルメニューを開き、Switch Workspace を選択してワークスペースランチャ（Workspace Launcher、p.2-21A）を開きます。
2. Browse ボタンを押して Select Workspace Directory ブラウザ（図 2-21B）を開き、切換え先となるワークスペースフォルダを選択します。TSB は一旦シャットダウンしてから指定の新しいワークスペースを使用して再起動します。

プロジェクト、スクリプトファイルの削除

プロジェクトの削除

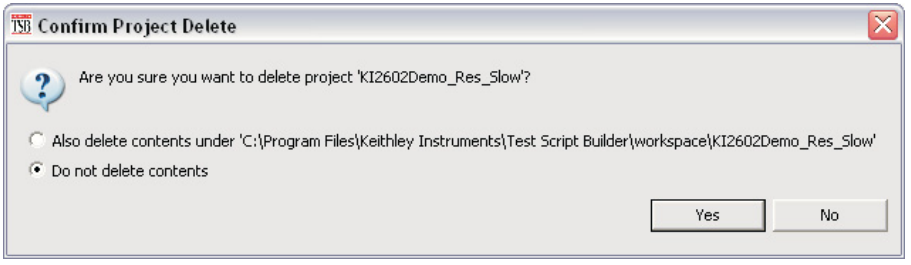
プロジェクトを削除するには、プロジェクトナビゲータの中のいずれかのプロジェクトを右クリックし、表示されたマウスメニューの中から Delete を選択して Confirm Project Delete ウィンドウ（図 2-23）を呼び出します。

プロジェクトを削除するために次の 2 つのオプションのいずれかを選択します。

- “Also delete contents under ... (プロジェクトへのパス)” – このオプションを選択すると、そのプロジェクトはプロジェクトナビゲータから削除されるばかりでなく、ファイルシステムのワークスペースフォルダからも削除されます。
- “Do not delete contents” – このオプションはプロジェクトをプロジェクトナビゲータからは削除しますが、ワークスペースフォルダからは削除しません。したがって、後でプロジェクトナビゲータへインポートして元へ戻すことも可能です（「別なワークスペースからプロジェクトをインポート」（p.2-36）参照）。

いずれかのオプションを選択してから、削除ウィンドウの Yes をクリックすることにより削除が実行されます。

図 2-23
プロジェクトの削除



スクリプトファイルはプロジェクトナビゲータから削除されるばかりでなく、そのプロジェクトのワークスペースフォルダからも削除されます。

スクリプトファイルの削除

プロジェクトからスクリプトファイルを削除したい場合は、プロジェクトナビゲータ上でスクリプトファイルを右クリックし、表示されるマウスメニューの Delete を選択します。スクリプトファイルはプロジェクトナビゲータから削除されるばかりでなく、そのプロジェクトのワークスペースフォルダからも削除されます。

コマンドと文の送信

ユーザ自身のプログラムまたは Test Script Builder を使用して、スクリプトではないチャンクを 1 回に 1 行ずつ実行することができます。これに対する応答（読み取り値）が PC へ返送されます。

電圧と電流の発生と測定

SMU の主たる機能は電圧または電流の発生源となり、それをもとに電流、電圧、抵抗、電力を測定することです。

次の部分プログラムでは smua を使用して測定電圧を発生させています。実測された電流と電圧読み取り値は PC へ送り返されます。

V を発生し、I と V を測定：

```
reset()                    -- ソースメータをデフォルト設定の状態に戻す
smua.source.levelv = 1     -- SMU A の V-source レベルを 1V にセット
smua.source.output = smua.OUTPUT_ON  -- 出力 ON
reading = smua.measure.iv()          -- I、V 測定実行
print(reading)                       -- PC が電流測定読み取り値を表示
smua.source.output = smua.OUTPUT_OFF -- 出力 OFF
```

デジタル I/O ポートの読み出しと書き込み

ソースメータはデジタル I/O ポートを使用して外部の回路をコントロールします（例えば、ビニング操作用の部品ハンドラ）。この I/O ポートは 14 の入出力ビット（ライン）を持ち、TTL 論理の 1（High）または 0（Low）の状態を取ります。デジタル I/O ポートのピン配列は図 10-1 に示されています。

ICL コマンドはこれらのポートをビット単位で読み出し / 書き込みするコマンドと、ポート全体の読み出し / 書き込みを行うコマンドを提供しています。

例えば、次の部分コードを使用すればデジタル I/O ポートのあるビットに書き込みを行うことができます。次に I/O ビットを読み出し、その状態を PC へ返して表示させます。

```
digio.writebit(4,0)          -- I/O ビット 4 へ “0” を書き込む
data=digio.readbit(4)        -- I/O ビット 4 読み出し
print(data)                  -- PC が I/O ビット 4 の状態を表示
```

ユーザ定義メッセージの表示

オペレータはメッセージを自分で定義して、それをソースメータの前面パネルに表示させることができます。次の部分コードを実行すると “Test in Process” というメッセージがソースメータのディスプレイに表示されます。

```
display.clear                        -- ディスプレイ上のメッセージをクリア
display.settext("Test in Process")   -- メッセージを表示
```

オペレータに前面パネルからパラメータ値を入力させたい場合は、スクリプトの中でメッセージとプロンプト表示機能を使用するのが便利です。さらに詳しくは「対話スクリプト」（p.2-40）をご覧ください。

ユーザスクリプト

スクリプトを記述するには Test Script Builder ばかりでなくユーザの好みのプログラムも使用することができます。260x 型ソースメータへローディングしたユーザスクリプトは不揮発性メモリへ保存することも可能です。不揮発性メモリに保存した以外のスクリプトは 260x 型の電源が切れると失われます。

スクリプト例

コマンドと文のみを使用するスクリプト

表 2-1 に示すのは、電圧スイープ（1V ～ 5V）をかけて各ステップでの電流を測定するスクリプトです。5 個の電流読み取り値はホストコンピュータへ送られます。

表 2-1
電圧をスイープし、電流を測定するスクリプト例

Test Script Builder	ユーザのプログラムスクリプト
current = {} smua.source.output = smua.OUTPUT_ON for j = 1, 5 do smua.source.levelv = j current[j] = smua.measure.i() print(current[j]) end smua.source.output = smua.OUTPUT_OFF	loadscript current = {} smua.source.output = smua.OUTPUT_ON for j = 1, 5 do smua.source.levelv = j current[j] = smua.measure.i() print(current[j]) end smua.source.output = smua.OUTPUT_OFF endscript

注 **Test Script Builder を使用してスクリプトを作成する場合は上に示すようにチャンクの本体だけを入力すれば充分です。スクリプトの作成とローディング、および実行の詳細については「Test Script Builder の使用法」（p.2-12）をご覧ください。**
プログラミング言語を使用してスクリプトを作成するのであれば、ホストコンピュータと TSP 間の連携をとるためにシェルコマンドを追加する必要があります。loadscript コマンドがスクリプトを 260x 型へローディングし、endscript がスクリプトの終了を通知します。

関数を使用するスクリプト

TSL は関数キーワードを用いてコマンドや文をグループ化する機能を備えています。したがって、スクリプトが 1 つまたは複数の関数によって構成されることもあります。スクリプトが一旦実行（RUN）されると、それ以後はホストコンピュータからスクリプト内の関数を直接呼び出せるようになります。

表 2-2 に示すスクリプトは測定速度（NPLC）を設定する ICL コマンドと関数（sourcev という名前）を含んでおり、このスクリプトが実行されると測定速度が 0.5 PLC されるとともに関数 sourcev が呼び出し可能な状態になります。

表 2-2
関数を使用するスクリプト例

Test Script Builder	ユーザのプログラムスクリプト
smua.measure.nplc = 0.5 function sourcev(v) smua.source.levelv = v i = smua.measure.i() print(i) return(i) end	loadscript smua.measure.nplc = 0.5 function sourcev(v) smua.source.levelv = v i = smua.measure.i() print(i) return(i) end endscript

この関数を呼び出すときは、ソース電圧を関数への引数として与えなければなりません。例えば、ソースを 2 V にセットしたい場合の関数呼び出しは次のようになります。

```
sourcev(2)
```

SMU A の出力がオンになっていれば、装置は 2V を発生して電流を測定します。電流読み取り値はホスト PC へ送られて表示されます。

対話スクリプト

対話スクリプトはオペレータに対してプロンプト（ソースメータのディスプレイ）を表示して、試験パラメータ入力（ソースメータの前面パネルへ）を要求します。表 2-3 に例を示す部分チャンクではメッセージ表示を利用してオペレータに SMU チャンネル（A、または B）、ソース機能（I、または V）を選択させ、ソースレベルを入力させています。スクリプトは入力プロンプトを表示し、オペレータがパラメータ値を入力するか、または ENTER キーを押すまで待ち続けます。

次のスクリプトでは display.prompt コマンドを使用してユーザにソールレベルを入力させています。値が入力されなかった場合は、ENTER が押されると同時にデフォルトレベル（1mA または 1V）が設定されます。また、このスクリプトではオペレータが許容上下限（下限：0.5mA/0.1V、上限：3mA/10V）を超える値を入力できないようにプログラムされています。

表 2-3
スクリプトの中で対話操作を行う部分の例

スクリプトチャンクのフラグメント
（Test Script Builder またはユーザプログラム）

```
-- プロンプトを表示してオペレータにチャンネルを選択させます。
 chan = display.menu ("Select Channel", "smua smub")
 if (chan == "smua") then
  chan = smua
 end
 if (chan == "smub") then
  chan = smub
 end

-- プロンプトを表示してオペレータにソース機能を選択（入力）させます。
 func = display.menu("Select Function", "amps volts")
 if (func == "amps") then
  chan.source.func = chan.OUTPUT_DCAMPS
 else chan.source.func = chan.OUTPUT_DCVOLTS
 end

-- プロンプトを表示してソースレベルを設定（入力）させます。
 if (func == "amps") then
  level = display.prompt("0.0E+00"," mA", "Enter I level",
      1E-  3, 0.5E-3, 5E-3)
 else
  level = display.prompt("00.0", " V", "Enter V level",
      1,   0.1, 10)
 end

-- オペレータがソースレベルを設定するのを待ちます。
 if (func == "amps") then
  chan.source.leveli = level
 else chan.source.levelv = level
 end
```

ユーザスクリプトの作成

スクリプトを作成してローディングするためには、テストプログラム（チャンク）を次のシェルコマンドで囲む必要があります。loadscript / loadandrunscript、および endscript。

ローディングのみ – 次のスクリプトは 260x 型のランタイム環境にローティングされますが、実行はされません。左のスクリプトには名前がないのに対して、右のスクリプトでは名前（ユーザが定義した名前）が指定されていることに注意してください。

```
loadscript          loadscript name
    (チャンク)          (チャンク)
endscript           endscript
```

ローディングと実行 – 次のスクリプトはランタイム環境へのローディングに続いて実行されます。スクリプトの実行で実際に実行されるのはチャンクのみです。この事実を常に忘れないでください。左のスクリプトには名前がないのに対して、右のスクリプトでは名前（ユーザが定義した名前）が指定されていることに注意してください。

```
loadandrunscript    loadandrunscript name
    (チャンク)          (チャンク)
endscript           endscript
```

loadscript および loadandrunscript コマンドの詳細を以下に説明します。

loadscript
loadscript name

ここに、name はユーザがそのスクリプトに付けた名前です。

シェルコマンド loadscript はスクリプトをランタイム環境にローディングします。このスクリプトは名前を指定することも、名無しのままにしておくことも可能です。すでにローディング済みのスクリプトと同じ名前を指定すると、古いスクリプトは新しいスクリプトによって上書きされます。

名前を与えずにランタイム環境にローディングしたスクリプトは、別なスクリプトのローディングによって失われ、また 260x 型の電源が切れると失われます。名無しでローディングされたスクリプトを実行するには run() または script.run() コマンドを使用します。

autoexec という特別なスクリプト名が用意されています。スクリプトを autoexec という名前で不揮発性メモリに保存しておくと、260x 型への電源投入後すべてのオートランスクリプトが実行された後で、この autoexec が自動的に実行されます。詳細については「Autoexec スクリプト」（p.2-45）と「オートランスクリプト」（p.2-44）をご覧ください。

loadandrunscript
loadsandruncript name

ここに、name はユーザがそのスクリプトに付けた名前です。

これらのコマンドは、スクリプトのランタイム環境へのローディング後に実行を伴うことを除けば loadscript コマンドと良く似ています。また、名前付きスクリプトの自動実行属性は “yes” に設定されます（「オートランスクリプト」（p.2-44）参照）。

ユーザスクリプトの保存

作成され、260x 型へローディングされたスクリプトは必ずしも不揮発性メモリに保存されるとは限りません。不揮発性メモリに保存しなかったスクリプトは 260x 型の電源が切れると同時に失われます。

名前付きスクリプトの保存

260x 型の不揮発性メモリに保存できるのは名前付きスクリプトだけです。名前付きスクリプトを作成してローディングしたならば、次のいずれかのコマンドを使用して保存します。

```
myscript.save()
myscript.save("name")
```

ここに、myscript はユーザが定義したスクリプト名であり、name はそのスクリプトに対してユーザが割り付けた新しい名前です。
この関数の機能は、Test Script Builder の File メニューの中にある “Save As” に相当します。

上記のどちらのコマンドを使用してもスクリプトが不揮発性メモリに保存されます。不揮発性メモリに保存されなかったスクリプトは 260x 型の電源が切れると同時に失われます。

myscript.save() コマンドは、そのスクリプトを作成してローディングした時のオリジナルの名前で保存します。それに対して、myscript.save("name") シェルコマンドはスクリプトを別名で保存します。不揮発性メモリ上にすでに存在する名前を使用してスクリプトを保存すると、古いスクリプトが上書きされます。

例：

1. “test1” という名前のスクリプトが作成され、ローディングされているとすれば、次のコマンドはそのスクリプトを同じ名前で不揮発性メモリに保存します。
   ```
   test1.save()
   ```
2. “test1” という名前のスクリプトを別名（“test2”）で不揮発性メモリに保存するのであれば次のコマンドを送信します。
   ```
   test1.save(test2)
   ```

ユーザスクリプトの実行

名無しスクリプトの実行

ランタイム環境に存在できる名無しスクリプトは 1 つに限られます。別な名無しスクリプトを作成してローディングすると、それ以前に存在した名無しスクリプトはランタイム環境から削除されます。最後にローディングされた名無しスクリプトのチャンクを実行するには、次のいずれかのコマンドを使用します。実行する内容はどちらのコマンドも同じです。

```
run()
script.run()
```

名前付きスクリプトの実行

ランタイム環境に存在する名前付きスクリプトを実行するには次のいずれかのコマンドを使用します。実行する内容はどちらのコマンドも同じです。

myscript()
myscript.run()

ここに、myscript はユーザが定義したスクリプト名です。

例：

スクリプトが “test3” という名前でランタイム環境にローディングされているとすれば、次のコマンドによってそのスクリプトのチャンクが実行されます。

```
test3()
```

スクリプトの自動実行

260x 型に電源が入ったときにスクリプトが自動的に実行されるように設定することができます。1 つまたは複数のスクリプトをオートランとして、そして 1 つを autoexec として指定することができます。

オートランスクリプト

保存済みのスクリプトをオートランとして設定すると、そのスクリプトは 260x 型に電源が入ると自動的にローディングされて実行されます。任意の数のスクリプトにオートラン属性を設定できますが、その実行については特定の順序を指定することができません。したがって、実行順序が問題とはならないスクリプトでなければなりません。

あるスクリプトをオートランに設定するには次に説明するオートラン（autorun）属性を “yes” に設定します。この属性として “no” を指定するとオートラン特性が解除されます。

myscript.autorun

ここに、myscript はユーザが定義したスクリプト名です。

autorun 属性設定後のスクリプトは必ず不揮発性メモリに保存してください。

例：

ランタイム環境に “test5” という名前のスクリプトが存在すると仮定します。このスクリプトをオートランにするには次のように設定します。

```
test5.autorun = "yes"
test5.save()
```

次回 260x 型に電源を入れると、“test5” スクリプトが自動的にローディングされて実行されます。

注 loadandrunscript name コマンドを実行するとそのスクリプトの autorun 属性が “yes” に設定されます。オートランを解除するには、autorun 属性を “no” にセットしてから保存してください。

Autoexec スクリプト

1 つのスクリプトだけを autoexec という特別なスクリプトに指定することができます。260x 型に電源が入り、すべてのオートランスクリプトの実行が終了すると autoexec スクリプトがスタートします。

```
loadscript autoexec
loadandrunscript autoexec
```

新しいスクリプトを作成してそれを autoexec という名前で保存することにより autoexec スクリプトが作られます（上に示すように loadscript または loadandrunscript を使用します）。新しいスクリプトのローディング後に autoexec.save() コマンドを送って、そのスクリプトを不揮発性メモリに保存してください。スクリプトの作成法の詳細については「ユーザスクリプトの作成」（p.2-42）の説明を参照してください。

不揮発性メモリに保存されている作成済みスクリプトの名前を付け替えて autoexec スクリプトにすることも可能です。そのためには次のコマンドを実行します。

```
myscript.save("autoexec")
```

ここに、myscript はユーザが定義したスクリプト名です。

例：

スクリプトが “test6” という名前で不揮発性メモリに保存されているとすれば、次の方法でそれを autoexec スクリプトに変えることができます。

```
test6.save("autoexec")
```

次回 260x 型に電源が入り、すべてのオートランスクリプトの実行が終了すると、この “test6” スクリプトが自動的にローディングされて実行されます。

260x 型の前面パネルを操作してユーザスクリプトを実行する

前面パネルから操作してユーザスクリプトを実行するためには、LOAD キーで呼び出す User メニューにそのスクリプトを指定する項目を追加する必要があります。User メニューに名前を追加 / 削除するには以下のコマンドを使用します。

```
display.loadmenu.add(displayname, script)
display.loadmenu.delete(displayname)
```

ここに、displayname は User メニューに追加 / 削除される名前、script はそのスクリプトの名前です。

User メニューが選択されると、メニュー項目は自動的にアルファベット順に並べられて表示されます。したがって、User メニューに名前を追加する順序は問題ではありません。

例：

ユーザスクリプトが “Test 9” という名前でランタイム環境にローディングされているとすれば、次の方法でその名前（“Test9”）を User メニューに追加することができます。

```
display.loadmenu.add(Test9, Test9)
```

User メニューに名前を追加しておけば、前面パネルを次のように操作してスクリプトを実行することができます。

1. LOAD キーを押します。
2. User を選択します。
3. 実行したいスクリプトを選択して RUN キーを押します。

ユーザスクリプトの変更

不揮発性メモリに保存されたユーザスクリプトは、そのスクリプトのリストを呼び戻して変更することができます。呼び戻したスクリプトに変更を施したならば、再度ローディングして不揮発性メモリに保存します。詳細については「ユーザスクリプトのリスティング取り出し」（p.2-46）の説明をご覧ください。

注　不揮発性メモリに保存されたユーザスクリプトを Test Script Builder を使用して変更するには、プロジェクトナビゲータからスクリプトのリスティングを読み出してください（「260x 型装置からスクリプトを読み出す」、p.2-26、参照）。

スクリプトの管理

ユーザスクリプトのリスティング取り出し

不揮発性メモリに保存されているユーザスクリプトのリスティングを読み出すことができます。読み出したスクリプトの変更後は、同じ名前または別名のユーザスクリプトとして再び保存します。

注　変更後のユーザスクリプトは同じ名前で、または名前を変えて再び 260x 型へローディングしてください。

次のコマンドを実行すると 260x 型に保存されているユーザスクリプトの一覧が返されます。

`script.user.catalog``()`

例：

ユーザスクリプト一覧を取り出します。

```
for name in script.user.catalog() do
    print (name)
end
```

次の関数はスクリプトを取り出し、スクリプトのチャンクとシェルキーワード（loadscript / loadandrunscript、および endscript）を返します。

myscript.list()

ここに、myscript はユーザが定義したスクリプト名です。

例：

“test7” という名前で保存されているスクリプトのリスティングを取り出します。

```
userscriptlist = test7.list()
print (userscriptlist)
```

不揮発性メモリからスクリプトを削除

ランタイム環境のスクリプトに交換 / 変更 / 削除等の操作を行っても、不揮発性メモリに保存されたスクリプトは削除されません。不揮発性メモリからスクリプトを恒久的に削除するには以下のコマンドを使用します。

script.delete("name")
script.user.delete("name")

ここに、name はユーザが定義したスクリプト名です。

例：

“test8” という名前のスクリプトを不揮発性メモリから削除します。

```
script.delete("test8")
```

ランタイム環境にスクリプトを復元する

スクリプトは基本的にグローバル変数として扱われますから、新しい値を代入する、あるいは同名の新しいスクリプトをローディングすることによって置き換えられます。また、値 nil を割り付けることによってもランタイム環境から削除されます。不揮発性メモリからスクリプトを呼び出してランタイム環境に復元するには次のコマンドのいずれかを使用します。

script.restore("name")
script.user.restore("name")

ここに、name は復元の対象となるユーザ定義スクリプト名です。

例：

“test9” という名前のユーザスクリプトを不揮発性メモリから呼び出して復元します。

```
script.restore("test9")
```

ファクトリスクリプト

ファクトリスクリプトもユーザスクリプトと基本的に同じものですが、相違点は Keithley 自身が作成したスクリプトであり不揮発性メモリに恒久的に書き込まれていることです。ファクトリスクリプトに関するデータはセクション 13 にまとめられています。

「ユーザスクリプト」（p.2-39）で説明した事項の多くはファクトリスクリプトにもそのまま適用されますが、ファクトリスクリプトとユーザスクリプトの間には次のような相違点も存在します。

- ファクトリスクリプトは不揮発性メモリから削除できません。
- ファクトリスクリプトのリスティングを読み出して変更することは可能ですが、変更後のスクリプトはユーザスクリプトとして取り扱われます。ユーザスクリプトをファクトリスクリプトとして保存することはできません。

ファクトリスクリプトの実行

ファクトリスクリプトを実行するには次のいずれかのコマンドを使用します。

```
script.factory.scripts.name()
script.factory.scripts.name.run()
```

ここに、name はファクトリスクリプトの名前です。

例：

“sourceMeasureDC” という名前のファクトリスクリプトを実行します。

```
script.factory.scripts.sourceMeasureDC()
```

260x 型の前面パネルを操作してファクトリスクリプトを実行する

1. LOAD キーを押します。
2. Factory を選択します。
3. 実行したい関数を選択して RUN キーを押します。

ファクトリスクリプトの変更

ファクトリスクリプトのリスティング取り出し

ファクトリスクリプトのリスティングを読み出して変更することは可能ですが、変更後のスクリプトをファクトリスクリプトとして保存することはできません。変更後のスクリプトは同じ名前または別名のユーザスクリプトとして保存します。

注 **ファクトリスクリプトをインポートしてそれを 260x 型へ再度ローディングする場合は、ユーザスクリプトとしてのみ保存が可能です。**

次のコマンドを実行すると 260x 型に保存されているファクトリスクリプトの一覧が返されます。

`script.factory.catalog()`

例：

ファクトリスクリプト一覧を取り出します。

```
for name in script.factory.catalog() do
    print (name)
end
```

次の関数はスクリプトを取り出し、スクリプトのチャンクとシェルキーワード（loadscript / loadandrunscript、および endscript）を返します。

`script.factory.scripts.name.list()`

ここに、name はファクトリスクリプトの名前です。

例：

“sourceMeasureDC”という名前のファクトリスクリプトのリスティングを取り出します。

```
factoryscriptlist = script.factory.scripts.sourceMeasureDC()
print (factoryscriptlist)
```

相違点：リモート vs. ローカル状態

260x 型はローカルまたはリモート状態のいずれかにあります。装置はローカル状態（REM ランプ消灯）では前面パネルから操作され、リモート状態（REM ランプ点灯）にあるときは PC によってコントロールされます。起動直後の装置の状態はローカルです。

リモート状態

装置は以下の操作によってリモート状態に入ります。

- PC から装置へコマンドを送信する。
- 前面パネルからスクリプト（FACTORY または USER test）を実行する。この試験が完了すると装置は再びローカルモードへ戻ります。
- 装置と Test Script Builder 間の通信をオープンにする。

装置がリモート状態にある間は前面パネルからの操作が無効となりますが、LOCAL キーのみは（ロックされていない限りは）使用可能です。対話操作を伴うスクリプトの実行中は、オペレータがパラメータ値などを入力できるようにするため前面パネルからの操作が有効になります。

ローカル状態

以下の操作を実行するとリモート状態がキャンセルされてローカル状態へ復帰します。

- 装置の電源を切ってから再投入。
- 前面パネルの LOCAL キーを押す（キーがロックされていない場合）。
- PC から abort コマンドを送信する。
- Test Script Builder の Instrument Console に含まれるツールバーから Abort Execution アイコンを選択してクリックする。
- 前面パネルからスクリプト（FACTORY または USER test）を実行する。スクリプト完了後、装置はローカル状態に復帰します。

TSP- リンク システム

TSP- リンクを使用して試験システムを拡張すると、最高 64 台までの TSP- リンク対応装置を接続することができます。このシステムはスタンドアローンシステムであることも、PC でコントロールすることも可能です。TSP-Link を利用してシステムを拡張する方法の詳細はセクション 9 に解説されています。

スタンドアローンシステム – システム内の任意のノード（装置）の前面パネルからスクリプトを実行することができます。スクリプトを実行するとシステム内のすべてのノードがリモート動作（REM ランプ点灯）へ移行し、スクリプトを実行したノードがマスタ、それ以外のノードがスレーブとして機能します。スクリプトの実行が終了するとシステム内のすべてのノードがローカル動作に復帰（REM ランプ消灯）して、ノード間のマスタ / スレーブ関係も解除されます。

PC- ベースシステム – PC を使用するシステムでは、いずれか 1 つのノードへの GPIB/RS-232 接続がシステム全体へのインターフェイスとして機能します。GPIB または RS-232 インターフェイスを介してコマンドを送信すると、すべてのノードがリモート動作モード（REM ランプ点灯）へ入ります。

この動作モードではコマンドを受信したノードがマスタとなって、システム内の他のノード（スレーブ）をコントロールします。PC- ベースシステムでは、ノード間のマスタ / スレーブ関係を解除するためには abort コマンドの実行が必要となります。

メモリに関する注意事項（ランタイム環境）

260x 型はランタイム環境が動的に使用するために 8MB のメモリを確保しています。この中で、ファームウェアが通常のシステムの操作のために約 5MB のメモリを使用します。したがって、ユーザが使用できるのは残りの約 3MB です。これらのメモリ使用量は不揮発性メモリにローディングされたスクリプトの数によっても影響を受けます。上記の数値は、ファクトリスクリプトとユーザスクリプト用の不揮発性メモリがすべて埋まった状態で利用可能なメモリ量を表しています。

ランタイム環境とユーザが作成した読み込みバッファは、残されているこの 3MB の範囲に収まらなければなりません。これらの目的に使用するメモリ容量が 3MB を超えることもあり得ますが、これが起こるとメモリの割り付けエラーによってコマンドが予期の動作を実行できないという事態が発生する可能性があります。

メモリ割付エラーが発生すると、それ以後の装置の状態は保証されません。装置からデータをダウンロードしようとしてこのようなエラーに遭遇した場合は、装置の電源を一度切ってから再投入して装置を既知の状態に戻すようにお奨めします。電源を一度切ってから再投入すると、ランタイム環境とすべてのユーザ設定読み取りバッファがリセットされます。不揮発性メモリに保存していなかったデータはこの操作によって失われます。

ランタイム環境が使用しているメモリ量は関数 gcinfo を使用してチェックすることができます。gcinfo が返す最初の値がランタイム環境が現在使用しているメモリ量をキロバイト単位で表しています。ただし読み込みバッファが使用しているメモリはこの値に含まれていません。

読み込みバッファは要求した 1 つの読み取り項目ごとに約 15 バイトを消費します。その他に読み取りバッファ自体が使用する若干のオーバーヘッドが存在しますが、これはメモリ利用の計算に殆ど影響しない量です。例として、2 つの読み込みバッファの作成を想定し、一方が 1,000 個までの読み取り値、他方が 2,500 個までの読み取り値を保存できるものとします。この条件で読み込みバッファ用として予約されるメモリ量は次式で計算されます。

$(1000 \times 15) + (2500 \times 15) = 52, 500$ byte（52.5 KB）

不揮発性の読み込みバッファはランタイム環境が必要とするメモリを消費しないことに注意してください。したがって、不揮発性の読み込みバッファはメモリの必要量の計算に含めないでください。また、リモートノードの読み込みバッファはそのリモートノードのメモリは消費しますが、ローカルノードのメモリは消費しません。あるリモートノードはそのノード内で 3MB を超えないメモリを消費しますが、今問題にしているローカルノードのメモリ計算にそれを考慮する必要はありません。

TSL（Test Script Language）リファレンス

はじめに

スクリプトは TSP（Test Script Processor）が実行するプログラムであり、TSL（Test Script Language）を用いて記述されます。TSL は単純な構文と拡張性のあるセマンティクスを特徴とする効率の良い言語です。TSL はプログラム言語 Lua（Copyright c 1994-2004 Tecgraf、PUC-Rio）から派生した言語です。この言語について詳しくはプログラム言語 Lua の公式サイトである http:// www.lua.org をご覧ください。この他、Lua プログラム言語のユーザによって作られてメインテナンスされている http://lua-users.org も有用な情報源です。

予約語

```
and         function     return
elseif      nil          until
for         repeat       else
local       true         false
then        do           in
break       if           or
end         not          while
```

変数と変数型

TSL は nil、ブール型、数値、文字列、関数、およびテーブルという 6 つの基本変数型を備えています。TSL は動的に型の割り付けを行う言語ですから、変数の静的な型宣言を必要とせず、値が代入されたときにその値の性質に応じて型が決まります。したがって、値自体に型が付随します。まだ値が割り付けられていない変数は nil 型とみなされます。数値はすべて実数として扱われ、TSL では整数と浮動少数点数との区別はありません。

```
var = nil                                  -- var は nil 型です。
var = 1.0                                  -- var は数値になりました。
var = 0.3E-12                              -- まだ数値です。
var = 7                                    --  まだ数値です。
var = "Hello world!"                       -- var は文字列になりました。
var = "I said, Hello world!"               -- まだ文字列です。
var = function(a, b) return(a+b) end       -- var は関数になりました。
                                           -- この関数は 2 つの数値を加算します。
var = {1, 2., 3.00e0}                      -- var はテーブル（配列）になりました。
                                           3 個の初期化メンバーを持っています。
```

Nil は唯一つだけの値 nil を持つデータ型であり、他のすべての値とは異なる特性を持っています。グローバル変数は、値が代入されるまではデフォルトとして値 nil を持ち、グローバル変数を削除するときにも nil を代入します。TSL は nil を一種の値ではない値として用い、有用な値が存在しない状態を表します。

演算子

数値演算子：	関係演算子：	論理演算子：
+ (加算)	< (小なり)	and （論理積）
- (減算)	> (大なり)	or （論理和）
* (乗算)	<= (以下）	not (否定)
/ (除算)	>= (以上)	
- (符号反転)	~= (等しくない）	
	== (等しい)	

テーブル / 配列

TSL は “table”と呼ばれる、基本的に配列に似た非常に柔軟性に富むデータ型を多用します。

```
-- テーブルを定義します。
--
atable = {1, 2, 3, 4}-- 4 個の数値要素を持つテーブル。
-- テーブルを印刷します。
i = 1           -- テーブルのインデックスは 0 ではなく 1 から始まります。
-- atable[index] はそのインデックスに対応する値を持つときに True となります。
-- それ以外の場合は nil を返します。0 は False と評価されません。False となるのは nil だけです。
--
while atable[i] do
    print (atable[i]) -- インデックスを変化させてテーブルをスキャン。
    i = i + 1
end
```

上記コードの出力：

```
1
2
3
4
```

関数

TSL ではユーザが関数を定義することができます。関数は予め定義された数のパラメータを取り、複数のパラメータを返すように設計することができます。

```
-- 関数を定義して呼び出してみます。
--
function add_two(parameter1, parameter2)
    return(parameter1 + parameter2)
end
print(add_two(3, 4))
```

```
-- 関数定義に使用できる別なシンタックスを次に示します。関数は TSL におけ
-- る第 1 級値です。すなわち、関数は変数に保存することも、引数として渡す
-- ことも、処理結果として返すことも可能です。
add_three = function(parameter1, parameter2, parameter3)
        return(parameter1 + parameter2 + parameter3)
end
print(add_three(3, 4, 5))
```

```
-- 複数のパラメータを返す関数：引数として渡した 2 個の数値をもとに和、
-- 差、比率を返します。

--
function sum_diff_ratio(parameter1, parameter2)
    psum = parameter1 + parameter2
    pdif = parameter1 – parameter2
    prat = parameter1 / parameter2
    return psum, pdif, prat
end
sum, diff, ratio = sum_diff_ratio(2,3)
print(sum)
print(diff)
print(ratio)
```

上記コードの出力：

```
7
12
5
-1
0.66666
```

優先度

TSL の演算子は次の表に示す優先度を持ちます（上から下へ優先度が下がります)。

```
^
not    - ( 単項演算子 )
*      /
+      -
.. ( 連接演算子 )
<      > <=   >=     ~=     ==
and
or
```

‘^’ (指数) と ‘..’ が右結合性であることを除き、他の演算子はすべて左結合性です。したがって、下に示す左側の式は右側の式と等価です。

```
a+i < b/2+1            (a+i) < ((b/2)+1)
5+x^2*8                5+((x^2)*8)
a < y and y <= z       (a < y) and (y <= z)
-x^2                   -(x^2)
x^y^z                  x^(y^z)
```

論理演算子

論理演算子には and、or、not の 3 種類があります。制御構造がそうであるように、論理演算子も false と nil を偽と見なし、それ以外すべてを真と見なします。

演算子 and は判定結果が偽であれば最初の引数を返し、それ以外の場合は第 2 引数を返します。

演算子 or は判定結果が偽ではなかったときに最初の引数を返し、それ以外の場合は第 2 引数を返します。

```
print(4 and 5)
print(nil and 13)
print(false and 13)
print(4 or 5)
print(false or 5)
```

上記コードの出力：

```
5
nil
false
4
5
```

and と or は両者ともにショートカット評価を行います。すなわち、必要な場合以外は第 2 オペランドの評価を行いません。例えば、TSL の有用な構文である x = x or v は次の式と等価です。

```
if not x then x = v end
```

この式は、x の値が設定されていない場合に、x をデフォルト値 v にセットします（x は false ではないとします）。

数値 x、y のいずれか大きい方を選択したい場合には次の文を使用できます（and が or よりも高い優先度を持つことに注意）。

```
max = (x > y) and x or y
```

x > y が真の場合、and の最初の式が真ですから第 2 引数である x (これは数値ですから x も真) が式の結果となり、or を含む式の結果はその最初の式である x の値となります。x > y が偽であった場合は、and を含む式が偽となるため or 演算の結果は 2 番目の式である y を返します。

演算子 not は常に true または false を返します。

```
print(not nil)
print(not false)
print(not 0)
print(not not nil)
```

上記コードの出力：

```
true
true
false
false
```

連接

TSL は文字列の連接演算子を “..”（ドット 2 個）で表します。オペランドが数値であった場合、TSL はその数値を文字列に変換します。

```
print("Hello ".."World")
print(0 .. 1)
```

上記コードの出力：

```
Hello World
01
```

分岐

TSL の条件分岐はキーワード “if” を使用します。

```
--
------------------------------ IF ブロック -----------------------
--
if 0 then                          -- 数値のゼロは true であることに注意！
    print("Zero is true!")         -- C 言語では 0 が false と評価されますが、
else                                  TSL では nil 以外はすべて true です。
    print("Zero is false.")
end          --if 式 1

x = 1
y = 2
if (x and y) then
    print("' if ' expression 2 was not false.")
end      --if 式 2

if (x or y) then
    print("' if ' expression 3 was not false.")
end      -- if 式 3

if (not x) then
    print("' if ' expression 4 was not false.")
else
    print("' if ' expression 4 was false.")
end      -- if 式 4

if x == 10 then
    print("x = 10")
elseif y > 2 then
    print("y > 2")
else
    print("x is not equal to 10, and y is not less than 2.")
end      -- if 式 5
```

上記コードの出力：

```
Zero is true!
' if ' expression 2 was not false.
' if ' expression 3 was not false.
' if ' expression 4 was false.
x is not equal to 10, and y is not less than 2.
```

ループ制御

TSL は次のようなお馴染みの構文を用いて（規定回数、または式が false と評価されるまで）反復処理を行います。

```
-- 反復処理の対象
list = {"One", "Two", "Three", "Four", "Five", "Six"}
--
------------------------------ FOR ループ --------------------------
--
print("Counting from one to three:")
for element = 1, 3 do
    print(element, list[element])
end
print("Counting from one to four,")
print("in steps of two:")
for element = 1, 4, 2 do
    print(element, list[element])
end
--
------------------------------ WHILE ループ ------------------------
--
print("Count elements in list")
print("on numeric index")
element = 1
while list[element] do      -- list [element] =nil となったときにループ脱出
    print(element, list[element])
    element = element + 1
end
--
------------------------------ REPEAT ループ -----------------------
--
print("Count elements in list")
print("using repeat")
element = 1
repeat
    print(element, list[element])
    element = element + 1
until not list[element]
```

上記コードの出力：

```
Counting from one to three:
1 One
2 Two
3 Three
```

```
1 から 4 まで、2 刻みで実行。
1 One
3 Three
index にしたがってリストの要素をカウント
1 One
2 Two
3 Three
4 Four
5 Five
6 Six
repeat 文を用いてリストの要素をカウント
1 One
2 Two
3 Three
4 Four
5 Five
6 Six
```

標準ライブラリ

上に説明した標準的なプログラム構文に加えて、TSL は文字列操作や数学演算へ有用な機能を提供する標準ライブラリを備えています。装置コントロール用の拡張ライブラリを備えているのも TSL の特徴であり、これらのライブラリが TSP から装置へアクセスするためのプログラムインターフェイスを提供します。これらのライブラリは TSP スタート時に自動的にローディングされますから、プログラマ自身が管理する必要はありません。

基本ライブラリ関数

`print(x)`	アクティブなホストインターフェイスへ引数 x を出力します。x が数値の場合は tostring() 関数を使用して文字列へ変換します。
`collectgarbage([limit])`	ガーベッジコレクションのしきい値を指定された値（KB 単位）に設定し、バイトカウンタと照合してチェックします。新しいしきい値がバイトカウンタの値よりも小さかった場合、TSL はガーベッジコレクションを即時実行します。リミット値を指定するパラメータを与えなかった場合はデフォルト値として 0 が使用されます（したがって、強制的にガーベッジコレクションサイクルが実行されます）。詳しくは「注」を参照。
`gcinfo()`	TSP が使用している動的メモリの量を KB 単位で返します。

tonumber(x [,base])	x を数値に変換して返します。x がすでに数値、または数値へ変換可能な文字列であれば数値を返し、それ以外の場合は nil を返します。 オプション引数は数値を解釈する際の基底を指定します。基底は 2 から 36 までの範囲であればどの整数でもかまいません。10 を超える基底を指定した場合は、文字 'A'（小文字も可）が 10、'B' が 11、という具体に 35 を表す 'Z' まで順に進みます。基底 10（デフォルト設定）の場合、数値はオプションである指数部に加えて小数部を持つこともあります。それ以外の基底は符号なし整数のみを受け付けます。
tostring(x)	あらゆるタイプの引数を受け取って、それを適切な形式の文字列に変換します。
type(v)	引数を 1 つだけ受け取り、その型を文字列で表現して返します。可能な返り値は "nil"、"number"、"boolean"、"table"、"function" のいずれかです。

注：TSL はメモリ管理を自動的に行いますから、新しいオブジェクトへのメモリの割り付けや使用済みオブジェクトのメモリの解放をユーザが心配する必要はありません。TSL は随時ガッベッジコレクタを実行し、死んだオブジェクト（もはや TSL からアクセスできなくなったオブジェクト）をかき集めることによって自動メモリ管理を行います。TSL のすべてのオブジェクト（テーブル、変数、関数、スレッド、文字列）が自動管理の対象となります。TSL は 2 種類の数値を使用してガーベッジコレクションのサイクルをコントロールします。その 1 つは TSL が使用している動的メモリを計数するカウンタであり、他の 1 つはしきい値です。カウンタのバイト数がこのしきい値を超えると、TSL がガーベッジコレクタを実行して死んだオブジェクトに属するメモリを回収します。これによりバイトカウンタが調節され、しきい値もカウンタの新しい値の 2 倍にリセットされます。

文字列関数ライブラリ

このライブラリは基本的な文字列操作（部分文字列の検索や抽出などを含む）に使用できる基本的な関数群を提供します。TSL における文字列のインデックス付けでは最初の文字位置が 1 になります（ANCI C では 0）。負のインデックスも可能であり、その場合は文字列末尾から逆方向へ付けられた番号と解釈されます。この場合も末尾の文字位置が 1 であり、そこから番号を減らしてゆきます。

string.byte(s [,i])	文字列中の i 番目の文字の内部数値コードを返します。インデックスが文字列の範囲を超えた場合は nil を返します。
string.char(i1, i1, …)	0 個以上の整数値を受け取り、引数の数に等しい長さの文字列を返します。返される文字列の個々の文字は対応する引数と同じ内部数値コードを持ちます。
string.format(fs, e1, e2, …)	可変数の引数を受け取り、それらを書式化して返します。書式化の形式は先頭引数（文字列）によって指定されます。文字列書式化の形式は ANCI C の print 関数ファミリーが使用する形式と同じです。唯一つの相違点はオプション / 修飾子 *、l、L、n、p、および h がサポートされていないことです。オプション c、d、E、e、f、g、G、I、o、u、X、および x はすべて数値引数を期待し、 s だけが文字列引数を期待します。%s で書式化される文字列値にゼロが埋め込まれていることは許されません。
string.len(s)	文字列の長さを返します。
string.lower(s)	与えられた文字列に含まれる大文字をすべて小文字に変換して返します。

string.rep(s, n)	文字列 s を n 個複製して連結した文字列を返します。
string.sub(s, i [,j])	文字列 s の i 番目から j 番目の文字からなる部分文字列を返します。i と j は負数であってもかまいません。引数 j が与えられなかった場合は -1 とみなされ、その場合は文字列の長さと同じと解釈されます。例えば、string.sub(s,1,j) は文字列の先頭から長さ j の部分文字列を返し、string.sub(s, -i) は末尾から長さ i の部分文字列を返します。
string.upper(s)	与えられた文字列に含まれる小文字をすべて大文字に変換して返します。

数学ライブラリ

このライブラリは ANCI C 数学ライブラリの大部分と同じインターフェイスを提供します。すべての三角関数はラジアンを使用して計算します。ラジアンと角度の相互変換には math.deg() と math.rad() 関数を使用します。

math.abs(x)	引数 x の絶対値を返します。
math.acos(x)	引数 x のアーク余弦関数を計算し、その主値を返します。
math.asin(x)	引数 x のアーク正弦関数を計算し、その主値を返します。
math.atan(x)	引数 x のアーク正接関数を計算し、その主値を返します。
math.atan2(y,x)	引数 y/x のアーク正接関数を計算し、その主値を返します。
math.ceil(x)	数学的に正確な整数値である引数 x を下回らない最小の浮動小数点数を返します。
math.cos(x)	引数 x の余弦を計算して返します。
math.deg(x)	引数 x（ラジアン）を角度に変換して返します。
math.exp(x)	引数 x の指数関数（ex）を返します。ここに e^x は自然対数の基底です。
math.floor(x)	数学的に正確な整数値である引数 x を超えない最大の浮動小数点数を返します。
math.log(x)	引数 x の自然対数を計算して返します。
math.log10(x)	引数 x の基底を 10 とする対数を計算して返します。
math.max(x, y, …)	数値引数の中の最大値を返します。
math.min(x, y, …)	引数の中の最小値を返します。
math.mod(x, y)	引数 x と同じ符号を持ち、絶対値が y の絶対値よりも小さく、かつ k*y+f = x を満たす k が存在するような f を計算して返します。
math.pi	π（3.141592654）の値を持つ変数。
math.pow(x, y)	x^y を返します。
math.rad(x)	引数 x（角度）をラジアンに変換して返します。
math.sin(x)	引数 x の正弦を計算して返します。
math.sqrt(x)	引数 x の負ではない平方根を返します。
math.tan(x)	引数 x の正接を計算して返します。

`math.frexp()`	引数 x を少数部 f と指数部 n に分けて両方の数値を返します：f,n = math.frexp(x)。このとき、f は 0.0 であるかまたは 0.5 <= | f | <= 1.0 を満たし、$f * 2^n$ は x に等しいものとします。
`math.ldexp(x, n)`	math.frexp() の逆数を返します。すなわち、$x * 2^n$ の値を計算します。
`math.random([x],[y])`	引数を与えずに呼び出すと [0, 1] の範囲の擬似乱数（実数）を返します。引数 x を与えて呼び出すと、[1,n] の範囲の擬似乱数（整数）を返します。2 つの引数 x, y を与えて呼び出すと [x, y] の範囲の擬似乱数（整数）を返します。
`math.randomseed(x)`	擬似乱数発生器のシード（seed）を設定します。シードが同じであると同じ並びの乱数列が発生します。

3
DUT 試験の接続

セクション 3 のトピックス

入出力端子ブロック , 3-2
入出力 LO と筐体の接地 , 3-3
センシング方法 , 3-5
  2 線式ローカルセンシング , 3-5
  4 線式リモートセンシング , 3-5
  センスモード選択 , 3-6
SMU の複数接続 , 3-7
ガードとシールド , 3-8
  ガード , 3-8
  ノイズシールド , 3-9
  安全シールド , 3-10
  シールドとガードの併用 , 3-11
テストフィクスチャ , 3-12
SMU をフローティングにする , 3-13
出力 OFF 状態 , 3-15

入出力端子ブロック

260x 型は DUT（Devices Under Test）との入出力接続のためにネジ端子ブロックを使用します。2602 型は 2 系統の端子ブロックを備え（図 3-1: SMU チャンネルごとに 1 個）、シングル SMU タイプの 2601 型は 1 系統の端子ブロックを備えています。

端子ブロックは 2 個の拘束ネジで固定されていますから、これらのネジを緩めることにより背面パネルから取り外すことができます。それぞれのネジ止め端子には 24 AWG（0.2mm^2）から 12 AWG（2.5mm^2）までの導線を接続することができます。

端子ブロックから DUT までの接続が終わってから、端子ブロックを背面パネルに取り付け拘束ネジをしっかりと固定してください。

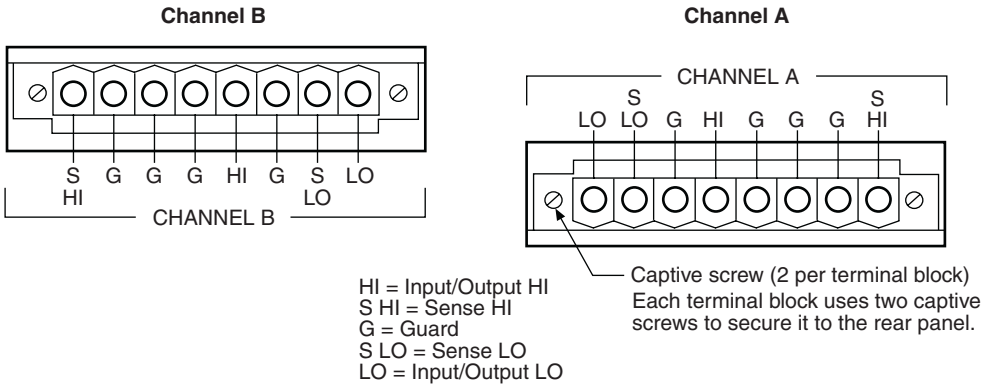
警告 **出力端子とガード端子には危険な高電圧が印加される可能性があります。感電事故防止のため、出力がオンの状態では絶対に 260x 型の接続や接続の切り離し作業を行わないでください。場合によっては致命的な傷害を引き起こす可能性があります。出力端子に接続されているケーブルを操作するときは、必ず事前に前面パネルから電源を切るか、またはソースメータ背面の電源ケーブルを引抜いてください。装置をスタンバイ状態に入れたとしても、ハードウェアやソフトウェアの障害によって出力が完全にオフになっていない可能性があります。**

SMU の最高浮動（コモンモード）電圧は 250V です。このレベルを超えると装置が破損して感電事故を引き起こす恐れがあります。SMU をフローティングにする方法の詳細については「SMU をフローティングにする」（p.3-14）をご覧ください。

ソースメータ入出力端子の定格は設置カテゴリー I 回路との接続のみに適合するように設計されています。したがって、ソースメータの端子を CAT II、CAT III、CAT IV 回路に接続しないでください。ソースメータの入出力端子を CAT I 規格を超える回路に接続すると装置の損傷を招き、オペレータを危険な電圧に曝す恐れがあります。

感電事故やソースメータの損傷を防止するため、260x 型以上の電流発生能力を持つソースを接続するときは、定格 3A を超えないヒューズをソースメータの入出力端子に沿って挿入してください。

図 3-1
入出力端子ブロック



入出力 LO と筐体の接地

図 3-2 に示すように、SMU の入出力 LO は背面パネルの端子ブロックに設けられています。入出力 LO はチャンネル間で接続されておらず、筐体 GND からも絶縁されています。

図に示されているように、低ノイズ筐体 GND がバナナジャックとして提供されていますから、これを入出力 LO の共通信号 GND として使用することができます。低ノイズ信号 GND を提供するこのバナナジャックは FVR（Frequency Variable Resistor）を介して筐体に接続されています。

この FVR（図 3-3 参照）は 260x 型筐体に重畳している可能性のある高周波から SMU を絶縁する機能を果たします。すなわち、筐体に重畳する周波数が高くなるにしたがって FVR の抵抗値も増大してノイズの干渉を防止します。

注　　筐体を信号接続の GND として使用しないでください。260x 型の筐体には周波数の高い信号が重畳している可能性がありますから、このような使い方は信号ノイズを高くする原因になります。筐体は安全シールドの目的のみに使用してください。260x 型筐体への接続には筐体ネジを使用してください。

図 3-2
入出力 LO と筐体接地端子

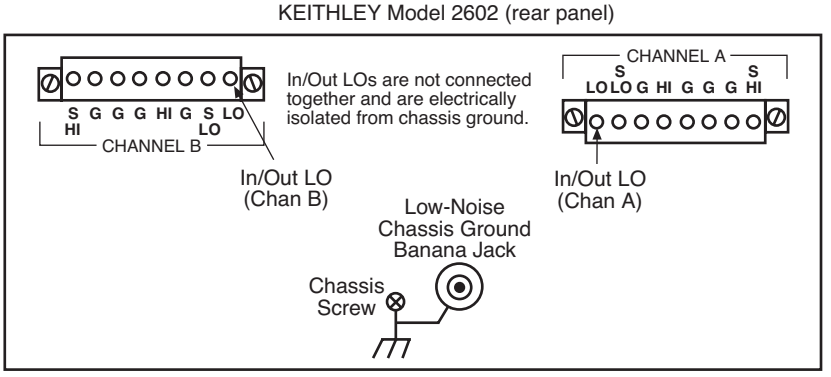
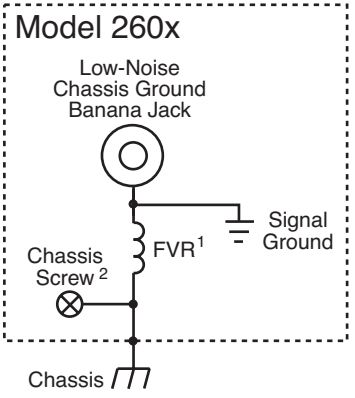

KEITHLEY Model 2602 (rear panel)
S G G G HI G S LO
HI LO
CHANNEL B
In/Out LOs are not connected together and are electrically isolated from chassis ground.
CHANNEL A
S S
LO LO G HI G G G HI
In/Out LO (Chan B)
In/Out LO (Chan A)
Low-Noise Chassis Ground Banana Jack
Chassis Screw

図 3-3
低ノイズ筐体 GND 用バナナジャックと筐体ネジ



1) Frequency Variable Resistor (FVR) Isolates the SMUs from high frequencies on the chassis. For DC to 60Hz, the FVR is a virtual short (zero ohms).
2) **DO NOT** use the Chassis Screw terminal to make signal connections to external circuitry. High frequency (>1MHz) on the chassis may result in higher noise at the output.



Signal Ground is a local signal ground and defined as the Low-Noise Chassis Ground Banana Jack.



Chassis is defined as the metal chassis of the Model 260x.



Chassis Screw terminal is connected to the metal chassis of the Model 260x.

センシング方法

ソースメジャー操作を実行するには、2 線式のローカルセンシング方式または 4 線式のリモートセンシング方式のいずれかの接続が用いられます。

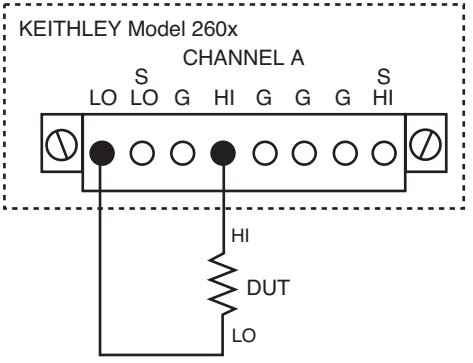
注　デフォルトでは 2 線式のローカルセンシングが選択されています。現在のセンスモードのチェック / 変更方法については「センスモード選択」（p.3-6）をご覧ください。

2 線式ローカルセンシング

2 線式ローカルセンシング（図 3-4 参照）は次のようなソースメジャー条件で用いられます。

- 電流を送り出して電流を測定する。
- 高インピーダンス（>1kΩ）試験回路の電圧測定、または電圧を印加しての電圧測定。

図 3-4
2 線式接続（ローカルセンシング）



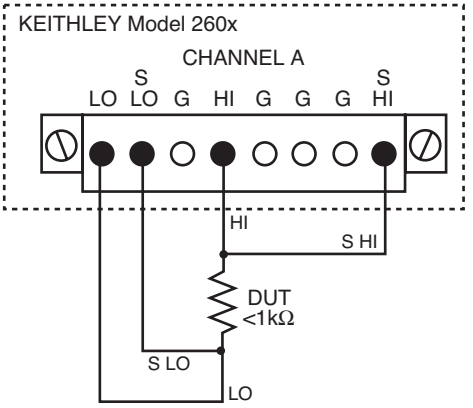
4 線式リモートセンシング

低インピーダンス試験回路（図 3-5）に正確にソースを印加して電圧を測ろうとすると、試験導線自体が持つ IR 抵抗によって測定誤差が生じます。このような場合には、4 線式のリモートセンス接続が電圧ソースと測定の確度を最適化する上で非常に有効です。4 線式リモートセンシング接続は実際にプログラムされた電圧を正確に DUT に送り出し、DUT 両端における電圧降下のみを正確に測定します。

以下のソースメジャー条件では 4 線式リモートセンシングを使用してください。

- 低インピーダンス（<1kΩ）試験回路の電圧測定、または電圧を印加しての電圧測定。
- 電圧コンプライアンスリミットを直接 DUT に課す測定。

図 3-5
4 線式接続（リモートセンシング）



センスモード選択

センスモードとして 2 線式ローカル、または 4 線式リモートのいずれかを選択します。

前面パネルからセンスモードを選択

前面パネルからセンスモードをチェック / 変更する手順をまとめて図 3-1 に示します。メニューへ入ったならば、回転ノブ（または CURSOR キー）を使用して点滅カーソルを希望するメニュー項目の上へ移動させ、ENTER キーを押してその項目を選択します。メニュー構造から脱出するには EXIT キーを押してください。

表 3-1
前面パネルからセンスモードを選択

2601 型	2602 型
1) CONFIG キーを押す	1) CONFIG キーを押す
2) SRC または MEAS メニューを選択 *	2) CHANNEL-A または CHANNEL-B を選択
3) V-SOURCE メニューを選択	3) SRC または MEAS メニューを選択 *
4) SENSE-MODE メニューを選択	4) V-SOURCE メニューを選択
5) 2WIRE または 4WIRE のいずれかを選択	5) SENSE-MODE メニューを選択
	6) 2WIRE または 4WIRE のいずれかを選択

* センスモードは SRC と MEAS のどちらのメニューからでも設定できます。

リモートプログラミングによるセンスモード選択

センスモードの選択に使用するコマンドをまとめて表 3-2 に示します。これらのコマンドの使用法について詳しくはセクション 12 をご覧ください。

表 3-2
センスモードを選択するコマンド

コマンド *	説明
smuX.source.output = smuX.OUTPUT_OFF	SMU 出力をオフにします。
smuX.sense = smuX.SENSE_LOCAL	ローカルセンシング（2 線式）を選択。
smuX.sense = smuX.SENSE_REMOTE	リモートセンシング（4 線式）を選択。

* 2601 型：smuX = smua　　2602 型：smuX = smua (Channel A) または smub (Channel B)

SMU の複数接続

図 3-6 に示すのは、例えば N- チャンネル JFET のような 3 端子デバイスの試験のために 2 台の SMU を接続する方法です。典型的な応用としては、SMU-B で一定範囲のゲート電圧を印加しながら SMU-A が電源電圧を供給し、各ステップのゲート電圧に対する電流を測定します。

図 3-6
3 端子デバイスに 2 台の SMU を接続（ローカルセンシング）

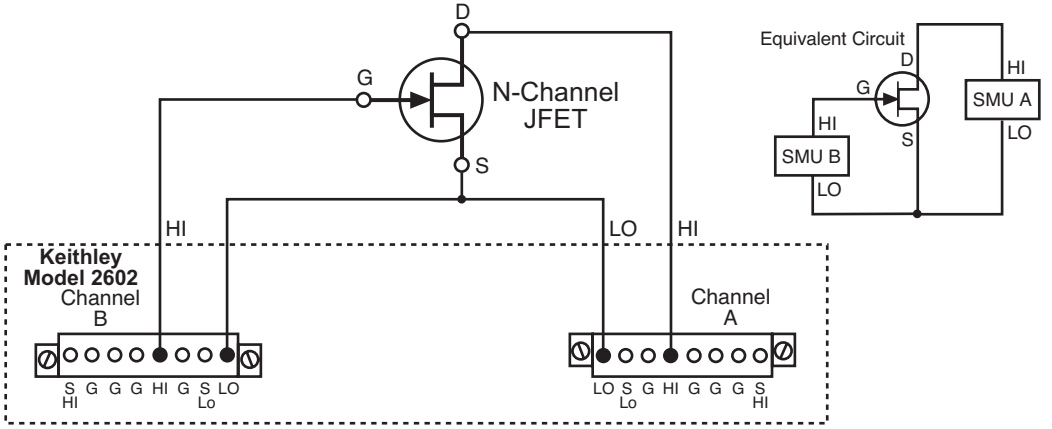
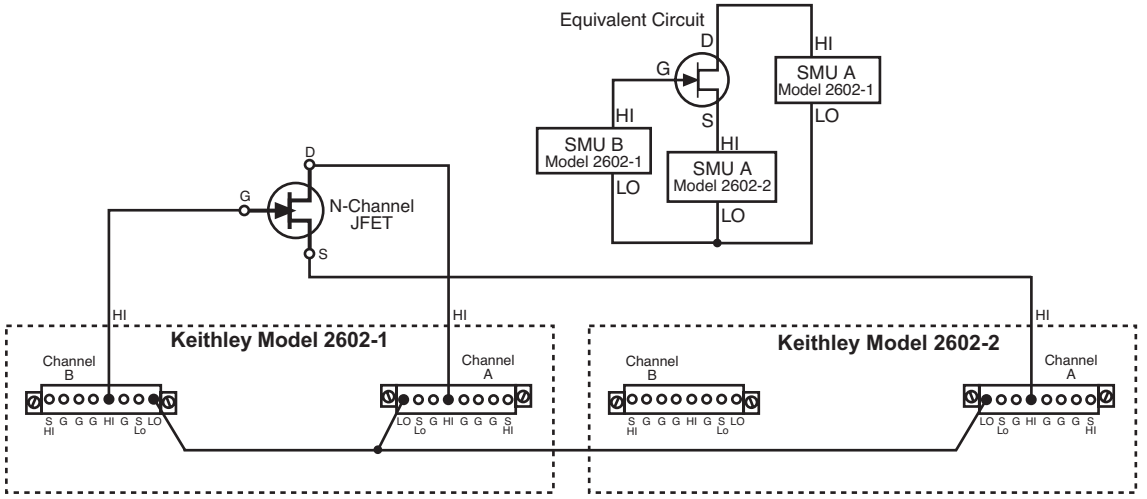


図 3-7 に示すのは同じ 3 端子デバイスの試験に 3 台の SMU を使用する構成です。この構成では 3 番目の SMU を JFET のソース端子（S）に接続することにより、ソース端子に対して信号の Low からかさ上げするバイアス電圧をかけることが可能になります。この SMU の出力を 0V に設定すれば、実質的に JFET のソース端子を信号の Low に接続したのと同じ効果が得られます。

図 3-7
3 端子デバイスに 3 台の SMU を接続（ローカルセンシング）



ガードとシールド

ソースメジャーで最高の性能と安全性を確保するためには、ガードとシールド（ノイズおよび安全性シールド）を効果的に使用することが重要です。

ガード

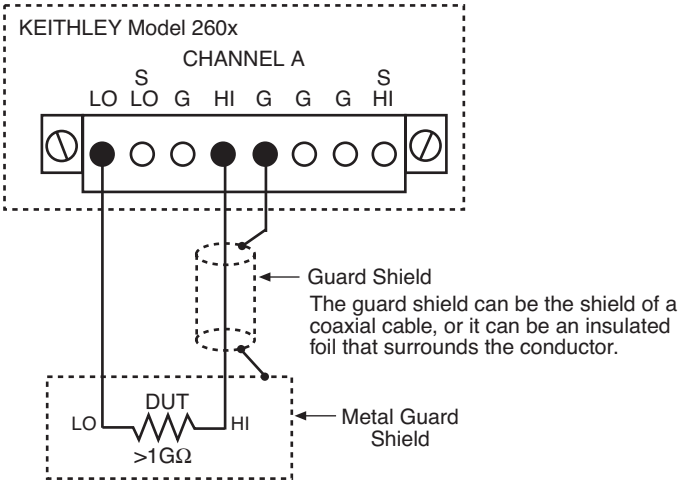
駆動付きガードはいつでも使用可能な状態にあり、入出力 HI 電圧と同じレベルのバッファ電圧を提供しています。ガードを付ける目的は、入出力 HI と LO 間の漏れ電流（およびキャパシタンス）の効果を打ち消すことにあります。ガードを使用しない場合は、外付け高インピーダンス試験回路の漏れ電流とキャパシタンスによって、ソースメータの性能が悪影響を受ける可能性があります。

以下のソースメジャー条件では必ずガード（図 3-8 参照）を使用してください。

- 試験回路インピーダンス >1GΩ

注 ガード使用の原則について詳しくはセクション 8「ガード」をご覧ください。

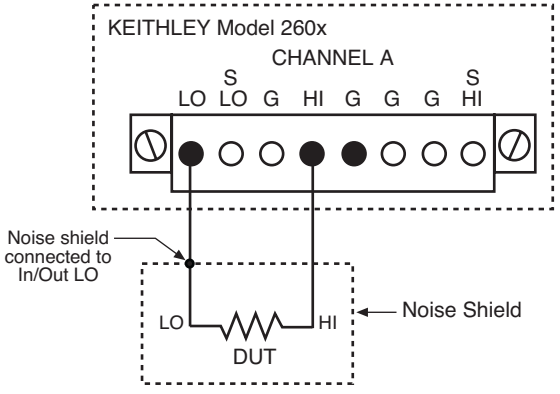
図 3-8
高インピーダンスガード



ノイズシールド

望ましくない信号の試験回路内への侵入を防止するためにノイズシールド（図 3-9 参照）を使用します。特に低レベルの信号を取扱う場合には効果的なシールドの実施が重要です。試験回路を金属製ノイズシールドで囲み、それを SMU LO（図 3-9 参照）へ接続してください。または、このシールドを 260x 型背面パネルの筐体 GND ネジ端子に落とすという方法も可能です。

図 3-9
ノイズシールド



安全シールド

試験回路に危険な電圧（>30Vrms、42Vpeak）が発生する場合は、必ず安全シールドを施さなければなりません。金属製と非金属製の安全シールドが考えられますが、どちらにしても DUT 試験回路を完全に覆う必要があります。金属製安全シールドは必ず既知の安全接地 GND や筐体 GND へ接続してください。（「テストフィクスチャ」（p.3-12）参照。この項には金属 / 非金属製のエンクロージャを使用する上での重要な安全情報が記載されています。）

260x 型 SMU の最大出力電圧は 40V ですから、それ自体では危険性のない電圧レベルと見なすことができます。しかし、2 台以上の電圧発生源を直列接続すれば試験回路に印加される電圧は簡単に 42V を超えてしまいます。例えば、2 台の 260x 型装置を直列接続（図 3-10）すれば DUT に 80V を印加することができます。

図 3-10 に示す試験構成の接続方法を図 3-11 に示します。安全のため、接地 GND と筐体の接続には #18AWG 以上の導線を使用してください。

注 **SMU の電位をフローティングにすることによって試験回路の電圧が 42V を超える可能性があります（「SMU をフローティングにする」（p.3-14）参照）。**

図 3-10
危険電圧（>42V）に対応する安全シールド

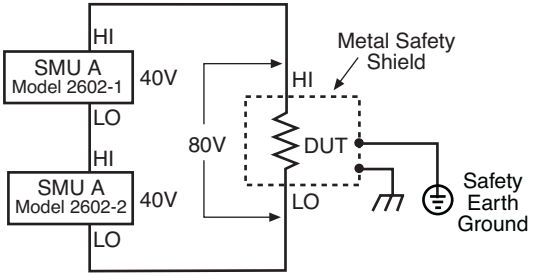
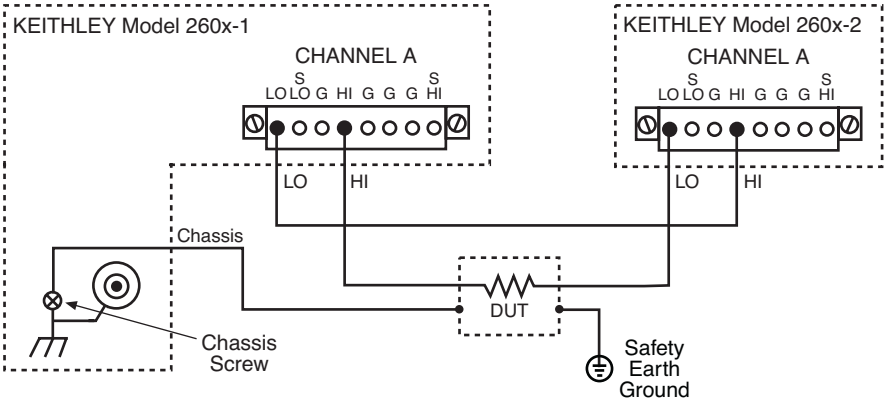


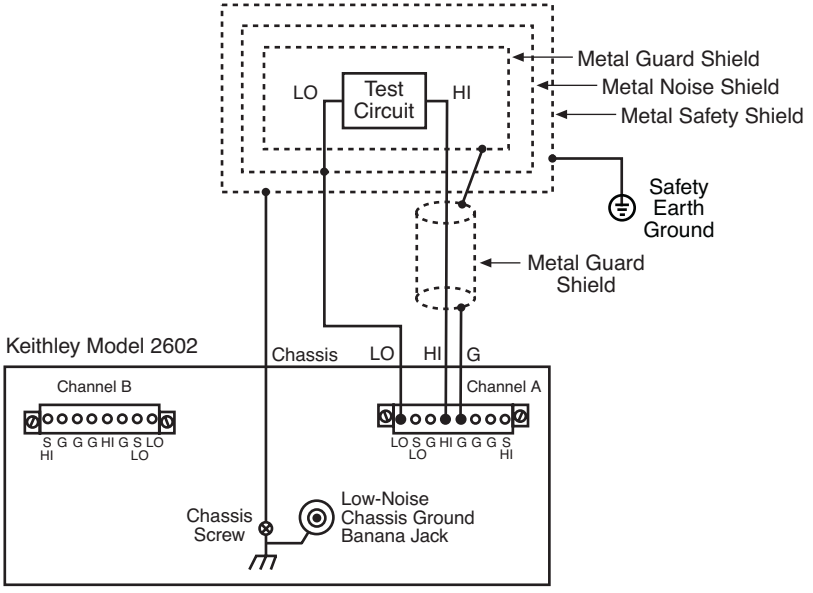
図 3-11
図 3-10 に示す試験構成の接続



シールドとガードの併用

図 3-12 に示すのはノイズシールドと安全シールド、およびガードを全部使用する試験システムの接続方法です。この例ではガードシールドが SMU の駆動付ガード（G）に接続されています。また、ノイズシールドは SMU LO に接続され、安全シールドは筐体に接続してから、更に安全接地 GND に接続されています。

図 3-12
ノイズシールド、安全シールド、およびガードの接続



テストフィクスチャ

外部の試験回路用に専用のテストフィクスチャを使用することがあります。テストフィクスチャは金属製の場合も非金属製の場合もありますが、多くの場合は蓋を備えており、試験回路をその内部に収納します。危険電圧（>30Vrms、42Vpeak）を使用する試験の場合、テストフィクスチャは次のような安全要件を満たしていなければなりません。

警告 **感電事故から保護するため、すべての通電部品を内部に納めるエンクロージャを備える必要があります。**

非金属製エンクロージャを使用する場合は、適切な難燃性を備え、試験回路の電圧と温度に適合した材料で組み立てる必要があります。

金属製エンクロージャの場合はテストフィクスチャ筐体を正しく安全接地 GND に接続する必要があります。安全接地用に設計されたテストフィクスチャ筐体のネジ端子に接地導線（#18 AWG 以上）を接続し、接地導線の他端を安全性が確認されている接地 GND へ落としてください。

作成に使用する材料 – 上の「警告」で説明したように、金属製テストフィクスチャは安全性の確認できた接地 GND へ接続しなければなりません。非金属製テストフィクスチャの場合は難燃性、耐電圧性、耐熱性の観点から試験回路の条件に適した材料で組み立てる必要があります。非金属製エンクロージャを組み立てる場合の要件についても「警告」で説明されています。

試験回路の絶縁 – テストフィクスチャは蓋を閉じた状態で完全に試験回路を隙間無く囲む必要があります。金属製テストフィクスチャは試験回路から電気的に絶縁されていなければなりません。金属製テストフィクスチャに取り付けてある入出力コネクタもテストフィクスチャからは電気的に絶縁されている必要があります。テストフィクスチャ内部では、標準的にはテフロン製スタンドオフを使用して試験回路のプリント基板やガードプレートを治具筐体から浮かせて電気的に絶縁します。

インターロックスイッチ – テストフィクスチャの蓋を開けると回路が自動的に遮断され、蓋を閉じると回路が接続されるようにするため、テストフィクスチャはノーマルオープンタイプのインターロックスイッチを備えている必要があります。

警告 **安全保持のためにインターロックが必要な場合は、アプリケーションの要求を満たしながらオペレータを危険な電圧から保護するための信頼できる回路を別に準備して頂く必要があります。2600 型シリーズのデジタル I/O ポートはこのような安全回路のコントロールには適しません。安全インターロックを制御する目的でこの I/O ポートを使用しないでください。**

SMU をフローティングにする

試験システムが外部ソースを使用するためには 260x 型の SMU を筐体接地 GND から浮かせなければならないことがあります。このような試験システムの例を図 3-13 に示します。この例では外部電圧ソースを使用しています。図 3-13 では電圧発生源の出力 Low が筐体接地 GND に接続されています。このような場合には 260x 型を筐体接地 GND から浮かせる必要があります。

図から分るように 260x 型 の SMU LO は筐体接地 GND から +10V 浮いています。この状態で 260x 型の SMU LO を筐体接地 GND へ接続すると、外部電圧ソースが筐体 GND を介して短絡状態になってしまいます。

図 3-13 に示す 260x 型の接続をフローティングにするのが図 3-14 に示す回路です。そのためには入出力 LO と筐体 GND の接続を取り外さなければなりません。

図 3-13、図 3-14 の外部電圧ソースを別な 260x 型装置の SMU で置き換えることも可能です。これらのソースを組み合わせた全体としての出力が 42V を超える場合には、DUT に安全シールドが必要となることに注意してください（次の「警告」参照）。

警告 **SMU の最高浮動（コモンモード）電圧は ± 250V です。このレベルを超えると装置が破損して感電事故を引き起こす恐れがあります。**

SMU をフローティングにするために使用した外部ソースによって試験回路内に感電の危険性が生ずることがあります。試験回路内に 42V を超える電圧が存在するときは常に感電の危険性が存在すると考えてください。すべての接続部に適切な定格のケーブルや絶縁体を用い、通電部分に直接手を触れることがないようにしてください。

42V を超える電圧が存在する場合は、試験回路に使用電圧に対応した絶縁を施すか、または金属製の安全シールドで囲い、それを特性既知の安全接地 GND と筐体 GND へ確実に落としてください（「安全シールド」（p.3-10）参照）。

図 3-13
260x 型をフローティングにする

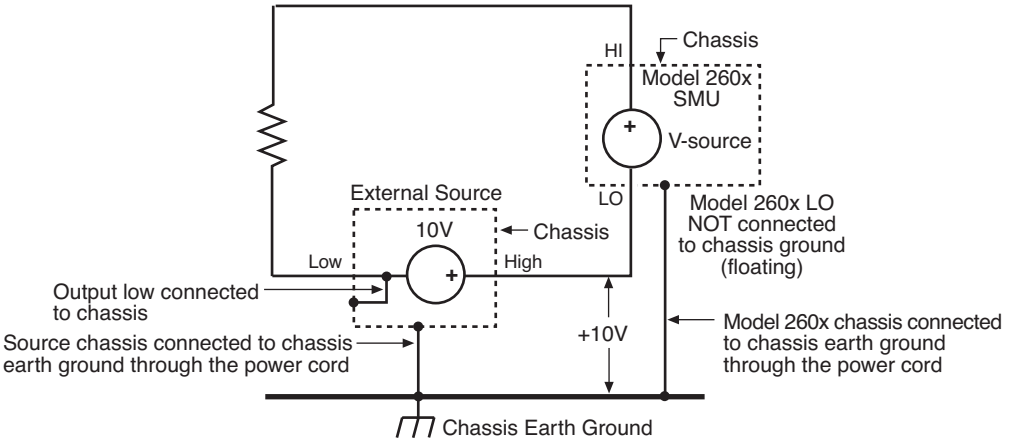
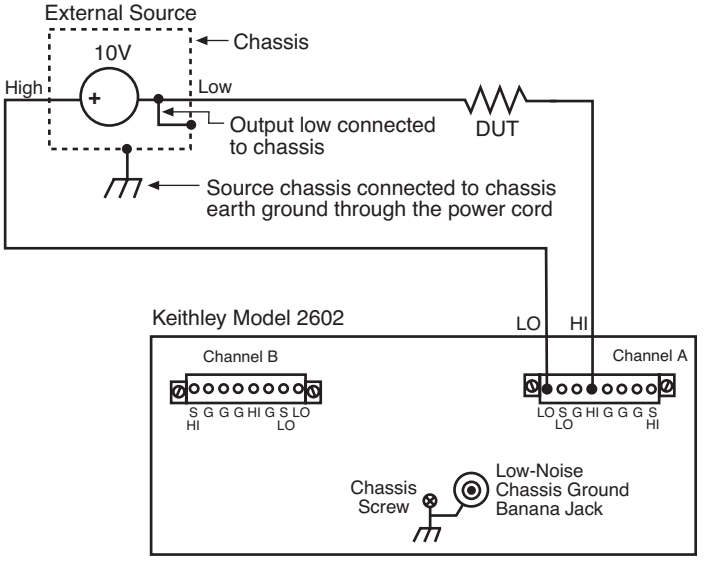


図 3-14
図 3-13 に示すフローティングの SMU 接続



出力 OFF 状態

SMU の電源をオフにしたとしても、SMU が外部回路から完全に切り離されるとは限りません。260x 型 SMU にはノーマル（Normal）、高インピーダンス（High Impedance）、およびゼロ（Zero）の 3 種類の出力状態が存在します。2602 型の場合にはそれぞれの SMU が独立してこれらの出力状態をとります。

出力 OFF 状態：ノーマル

出力 OFF 状態がノーマル（デフォルト設定）な場合、SMU は 0V を発生します。同時に電流コンプライアンスは現在の電流レンジ フルスケールの 10%、もしくは 100 μ A のいずれか値が小さい方に設定されます。SMU のソース値が 0V に設定されていれば、理論的には SMU によるパワーのソース / シンクは発生しないはずですが、実際にはソース値は厳密な意味でのゼロにはなりません。このため、極めて少量ですがパワーのソース / シンクが発生する可能性があります。ただし、殆どの場合この程度のソースやシンクは重要な問題にはなりません。

出力 OFF 状態：高インピーダンス

出力 OFF 状態として高インピーダンスを選択した場合、出力の電源を切ると出力リレーがオープン状態になり、外部回路を SMU の入出力から切り離します。出力リレーの余分な消耗を避けるために、出力の ON/OFF を頻繁に切換える必要のある試験では出力 OFF 状態として高インピーダンスを選択しないでください。

出力 OFF 状態：ゼロ（低インピーダンス）

出力 OFF 状態としてゼロを選択するとソースメータは次のように設定されます。

ソースとして V-Source を選択した場合：

- ディスプレイ上にはプログラムされた V-Source 値が表示されます。
- 内部では V-Source を 0V に設定します。
- 電流コンプライアンス設定値は出力 ON 状態の値を維持します。実際のコンプライアンス検出機能はアクティブなままです。
- 測定は継続して行われ、結果を表示します。

ソースとして I-Source を選択した場合：

- ディスプレイ上にはプログラムされた I-Source 値が表示されます。
- 内部的には V-Source が選択され、その値を 0V に設定します。
- 電流コンプライアンスはプログラムされた Source I 値、または現在の電流レンジフルスケールの 10% のいずれか大きい方の値に設定されます。
- 測定は継続して行われ、結果を表示します。

出力 OFF 状態としてゼロが選択されているときは、出力は 0V ですが電流測定は可能ですからソースメータを電流計として使用することができます。

出力 OFF 状態の選択

出力 OFF 状態メニュー

出力（OUTPUT）設定メニューへアクセスするには、まず CONFIG キーを押し、続いて OUTPUT ON/OFF キーを押します。設定メニューに入ったならば、OFF STATE を選択して OUTPUT OFF STATE メニューを表示させます。

注 CONFIG キーを押すと表示される設定メニューから順を追って OUTPUT OFF STATE メニューへアクセスすることも可能です。

OUTPUT OFF STATE メニューが表示されたならば、その中から希望する出力 OFF 状態を選択します。HI-Z (高インピーダンス)、NORMAL、または ZERO。

リモートプログラミング

出力 OFF 状態の選択に使用できるコマンドを表 3-3 に示します。

表 3-3
出力 OFF 状態を選択するコマンド

コマンド *	説明
smuX.source.offmode = smuX.OUTPUT_NORMAL	ノーマル出力 OFF 状態を選択
smuX.source.offmode = smuX.OUTPUT_HIGH_Z	高インピーダンス出力 OFF 状態を選択
smuX.source.offmode = smuX.OUTPUT_ZERO	ゼロ出力 OFF 状態を選択

* 2601 型：smuX = smua　　2602 型：smuX = smua (Channel A) または smub (Channel B)

4
基本操作

セクション 4 のトピックス

概要 , 4-2

操作のまとめ , 4-2
- ソースメジャー機能 , 4-2
- コンプライアンスリミット , 4-3
- コンプライアンスリミットの設定 , 4-4
- 基本回路構成 , 4-5

操作上の注意事項 , 4-7
- ウォームアップ , 4-7
- オートゼロ , 4-7
- NPLC キャッシング , 4-7

トリガ , 4-9
- トリガのタイプ , 4-9
- 測定トリガ , 4-10
- 前面パネルトリガ , 4-11
- リモートトリガ , 4-12

基本的なソースメジャー手順 , 4-13
- 前面パネルからソースメジャー手順を実行 , 4-13
- ソースメジャー手順のリモート操作 , 4-14

測定（メジャー）専用 , 4-16

シンク操作 , 4-17

抵抗測定 , 4-18
- 抵抗値の計算 , 4-18
- 抵抗測定のレンジ設定 , 4-18
- 基本的な抵抗測定操作 , 4-18
- 抵抗センシング , 4-19
- センシング方法の選択 , 4-20
- 抵抗測定のリモートプログラミング , 4-21

電力測定 , 4-22
- 電力の演算 , 4-22
- 電力測定の基本手順 , 4-22
- 電力測定のリモートプログラミング , 4-23

概要

このセクションでは基本的な操作方法を次の項目に分けて詳しく説明します。

- 「操作のまとめ」、p.4-2
- 「操作上の注意事項」、p.4-7
- 「トリガ」、p.4-9
- 「測定専用」、p.4-16
- 「シンク操作」、p.4-17
- 「抵抗測定」、p.4-18
- 「電力測定」、p.4-22

操作のまとめ

ソースメジャー機能

ソースメータが実行するべき以下の機能を前面パネルから選択することができます。

- **電圧ソース** — 電流測定値または電圧測定値（または両方）を表示します。
- **電流ソース** — 電圧測定値または電流測定値（または両方）を表示します。
- **抵抗測定** — 測定された電圧と電流から抵抗値を計算して表示します。
- **電力測定** — 測定された電圧と電流から電力を計算して表示します。
- **測定のみ（V または I）** — 電圧測定値または電流測定値を表示します。

電圧と電流

電圧と電流に関係する諸機能のソース発生と測定のリミット値一覧を表 4-1 に示します。

装置が動作可能な範囲（限界）について「動作の限界」（p.8-7）に説明があります。

> ***注　　測定に影響を与える性能上の重要事項が付録 A 「仕様」の項にまとめられています。***

表 4-1
ソースメジャー機能

レンジ	ソース	測定
100mV	±101mV	±102mV
1V	±1.01V	±1.02V
6V	±6.06V	±6.12V
40V	±40.4V	±40.8V
100nA	±101nA	±102nA
1µA	±1.01µA	±1.02µA
10µA	±10.1µA	±10.2µA
100µA	±101µA	±102µA
1mA	±1.01mA	±1.02mA
10mA	±10.1mA	±10.2mA
100mA	±101mA	±102mA
1A	±1.01A	±1.02A
3A	±3.03A	±3.06A
電力最大値 = 40.4W / チャンネル		

コンプライアンスリミット

ソースメータを電圧ソースとして使用する場合は電流にリミットを設定し、逆に電流ソースとして使用する場合は電圧にリミットを設定します。ソースメータ出力がコンプライアンスリミットを超えることはありません。最大コンプライアンスリミットは表 4-2 に列挙された最大値と同じ値を持ちます。コンプライアンス値の符号はソース値と一致し、最大コンプライアンスリミットはソースのレンジをもとにして決まります。詳しくは「コンプライアンスリミット」（p.8-2）の説明をご覧ください。コンプライアンスリミットを超えてはならないという規則の唯一つの例外は ISOURCE として動作させる場合の VLIMIT です。過大な（破壊を引き起こす可能性のある）電流が流れるのを防止するため、VLIMIT は ISOURCE の各レンジや 100mA 以下のレンジに対して 102mA までの電流を送り出す（ソース）または吸い込む（シンク）ことができます。1A またはそれを超えるレンジの場合は電流ソースの設定によって許容最大電流が決まります。

表 4-2
最大コンプライアンス値

ソースレンジ	最大コンプライアンス値
100mV	3A
1V	3A
6V	3A
40V	1A
100nA	40V
1µA	40V
10µA	40V
100µA	40V
1mA	40V
10mA	40V
100mA	40V
1A	40V
3A	6V

コンプライアンスリミットの設定

前面パネルからコンプライアンスリミットを設定

以下の手順に従って前面パネルからコンプライアンスリミットを設定します。

1. 2601 型または 2602 型のシングルチャンネル表示モードの場合は、LIMIT キーを押してコンプライアンス値の変更機能に直接アクセスします。
2. 2602 型のデュアルチャンネル表示モードの場合は、まず CONFIG を押してから LIMIT キーを押し、続いて CURRENT または VOLTAGE のいずれかを選択します。最後に ENTER キーまたは回転ノブを押してください。
3. 回転ノブを押し、コンプライアンスリミットを希望の値に設定し、ENTER または回転ノブを押して変更作業を完了します。
4. EXIT を押して通常表示に戻ります。

コンプライアンスリミットのリモート設定

コンプライアンスリミットのプログラムに使用する基本コマンドをまとめて表 4-3 に示します。これらのコマンドの使用法についてさらに詳しくはセクション 12 をご覧ください。これらのコマンドに希望のパラメータを与えて送信することによりコンプライアンスがプログラムされます。例えば、次のコマンドを実行すると電流と電圧のコンプライアンス値がそれぞれ 50mA と 4V に設定されます。

```
smua.source.limiti = 50e-3
smua.source.limitv = 4
```

次のコマンドはコンプライアンスの現在の状態を表示します。

```
print (smua.source.compliance)
```

装置を電流ソースとして設定しているときに値 1 が返されたとすれば電圧リミットに達したことを意味し、電圧ソースとして設定されているときに値 1 が返されたとすれば電流リミットに達したことを意味します。

表 4-3
コンプライアンスコマンド

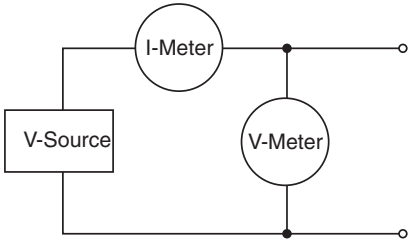
コマンド[1]	説明
smuX.source.limiti = limit smuX.source.limitv = limit compliance = smuX.source.compliance	電流コンプライアンスのリミット値を設定 電圧コンプライアンスのリミット値を設定 コンプライアンス状態にあるかテスト（1= コンプライアンス状態、0= 非コンプライアンス）

1. 2601 型：smuX = smua、2602 型：smuX = smua (Channel A) または smub (Channel B)

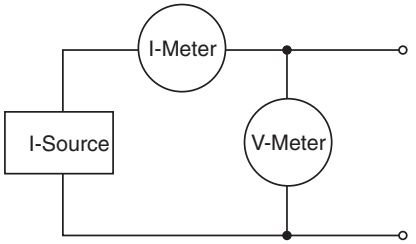
基本回路構成

ソースメータの基本的なソースメジャー構成を図 4-1 に示します。電圧を供給する設定では電流または電圧を測定し（構成 A）、電流を供給する構成の場合は電圧または電流を測定します（構成 B）。これらの回路構成について更に詳しくは「基本回路構成」（p.8-16）をご覧ください。

図 4-1
基本ソースメジャー構成



A. Source V



B. Source I

操作上の注意事項

以下の項ではウォームアップ期間とオートゼロについて解説します。

ウォームアップ

ソースメータは電源投入後少なくとも 2 時間のウォームアップ期間を置かなければ規定の確度を満たすことができません（仕様については付録 A を参照）。

オートゼロ

260x 型ソースメータはレシオメトリック方式の A/D 変換器を使用します。この方式は読み取り値の確度維持のために、定期的に内部 GND と基準電圧の値を取り込まなければなりません。これらの基準電圧測定が必要となる時間間隔は、測定のために現在使用している積分アパーチャによって決まり、毎回のアパーチャごとに基準値とゼロを分けて取り込みます。

オートゼロについては、表 4-4 を見ると分るように 3 種類の異なる設定が用意されています。デフォルト設定（AUTO）では、信号測定を行うときは必ず装置がこれらの基準電圧測定値を自動的にチェックし、信号測定を行う時点でこれらの基準測定値がすでに古くなっていると判断した場合は、変換結果を返す前に余分に 2 回の A/D 変換（基準電圧とゼロ）を実行します。このため、余分な A/D 変換を実行したサイクルの測定時間は標準よりも長くなります。

測定のタイミングが重要であるアプリケーションでは、この余分な変換時間がスイープその他の試験シーケンスに支障を及ぼす可能性があります。基準値測定に関わる余分な測定時間の発生を避けたい場合は、OFF を選択することによって基準値の自動測定を止めることができます。ただし、基準値の自動測定を使用しないと装置が徐々にドリフトして仕様の確度から外れて行く可能性があるので注意が必要です。

ドリフトを最小限に抑えるためには、少なくとも重要な試験シーケンスの実施前には必ず基準電圧とゼロの測定を行うべきです。現在のアパーチャ設定で基準電圧とゼロの測定値を強制的に更新させたい場合は、ONCE 設定を使用してください。

NPLC キャッシング

NPLC キャッシングは測定機能の設定で使用した A/D 基準値とゼロ値を最高で直近 5 回の設定分までキャッシングすることによって動作の高速化を図ります。SPEED キーを使用して積分速度を変更したりユーザセットアップを呼び出したときは、いつも NPLC のキャッシングが行われます。積分速度がすでにキャッシュに保存されている場合は、保存されている基準値とゼロ値を呼び戻して使用します。それ以外の場合は基準値とゼロ値を新たに取得してキャッシュに保存します。保存されている NPLC 値がすでに 5 個に達している場合は、一番古い値を新しい値で上書きします。

オートゼロが OFF の場合はキャッシュに保存されている NPLC 値を、それがどんなに古い値であっても使用します。現在使用中のアパーチャに対応する値がキャッシュに存在しない場合は、最初の測定実行時に必要な値を取り込みます。

表 4-4
オートゼロの設定項目

オートゼロ設定	説明
OFF	自動的に基準値を参照する測定を行いません。
ONCE	測定時に自動的に基準値を参照し、基準値とゼロ値の測定を必ず 1 回行います。
AUTO	プロセッサが基準値とゼロ値が古すぎると判断した場合に自動的に新しい値を取り込みます。

前面パネルからオートゼロを実行

以下の手順に従って前面パネルからオートゼロを設定します。

1. CONFIG キーを押し、表示されるメニューの中から MEAS を選択します。
2. AUTO-ZERO を選択し、ENTER または回転ノブを押します。
3. 希望するモード（OFF/ ONCE/ AUTO）を選択し、ENTER または回転ノブを押します。
4. 通常表示へ戻るには EXIT を押します。

リモートコマンドを使用するオートゼロ

表 4-5 に示すオートゼロコマンドに適切なオプションを与えることにより、オートゼロをリモート設定することができます（更に詳しくはセクション 12 を参照）。例えば、次のコマンドを送ると自動基準値測定がオンになります。

```
smua.measure.autozero = smua.AUTOZERO_AUTO
```

表 4-5
オートゼロコマンドとオプション

コマンド[1]	説明
smuX.measure.autozero = smuX.AUTOZERO_OFF	オートゼロをオフにします。[2]
smuX.measure.autozero = smuX.AUTOZERO_ONCE	強制的に基準値とゼロ値を 1 回取り込みます。
smuX.measure.autozero = smuX.AUTOZERO_AUTO	毎回の測定で基準値とゼロを強制的に取り込みます。

1. 2601 型：smuX = smua、2602 型：smuX = smua (Channel A) または smub (Channel B)
2. オートゼロ機能をオフにすると古い NPLC キャッシュ値を使用します（「NPLC キャッシング」（p.4-7）参照）。

トリガ

トリガのタイプ

トリガを与えることにより 260x 型ソースメータ内でイベントが始まります。一般的にはトリガを発生させる方法として次の 3 種類があります。

- **測定トリガ** — 1 回以上の測定をスタートさせるとき、測定から測定までの時間またはトリガ発生から測定までの時間をコントロールするとき、および 1 回のトリガで実行する測定回数を設定するときにこのタイプのトリガを使用します。詳しくは下の「測定トリガ」の項を参照。
- **デジタル I/O ポートトリガ** — デジタル I/O ポートからパルスを発生して外部デバイスにトリガを与える場合、逆に外部からデジタル I/O ポートにコントロールパルスを入力して 260x 型にトリガをかけるときにこのタイプを使用します（詳しくはセクション 10 を参照）。
- **ディスプレイトリガ** — 前面パネルキーからイベントトリガを発生させる場合はこのタイプを使用します（セクション 14 参照）。

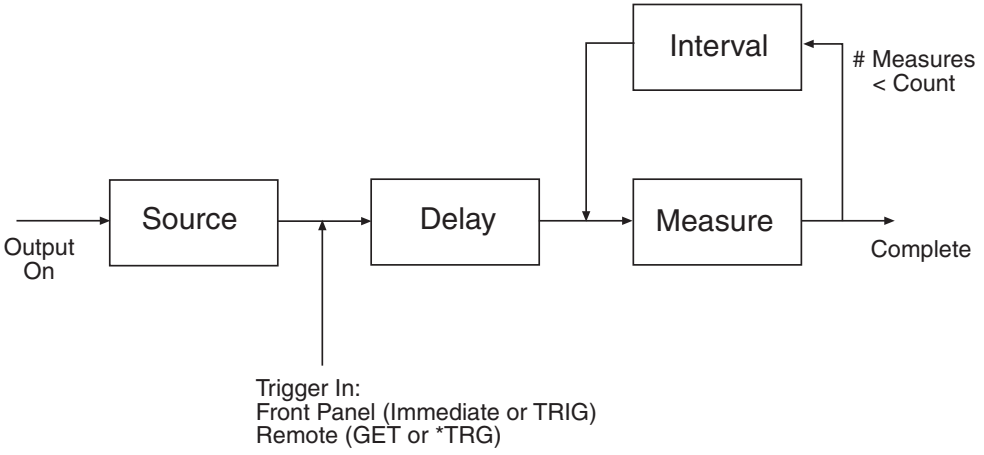
> ***注*** ***このセクションで説明する基本的なソーシングと測定の手順を使用する範囲ではトリガ発生に関わる設定を変更する必要はありません。ただし、操作を行う前に以下の順にメニューを選択して、単位が工場デフォルトの状態にリセットされていることを確認しておいてください。***
> ***MENU > SAVE-SETUP > RECALL > FACTORY。***

測定トリガ

測定トリガ発生の一般的なシーケンスを図 4-2 に示します。基本的な順序は次のようになります。

- 出力がオンになると同時にプログラムされているソース値が DUT に印加されます。
- 即時トリガソースが選択されている場合は、前面パネルからの操作によって直ちに測定トリガが発生します。ただし、手動トリガソースが選択されいる場合は前面パネルの TRIG キーを押さなければなりません。
- リモート操作の場合、装置は GET（GPIB のみ）または、* TRG（両インターフェイス）コマンドにプログラムされているタイムアウト時間の終了を待ちます。
- ディレイが設定されている場合、装置はその時間の終了を待ちます。
- 装置が 1 回の測定を実行します。
- 測定回数がプログラムされているトリガカウントに満たない間は、上記のサイクルを繰り返して測定を反復します（トリガカウントとして無限が選択されている場合は、測定サイクルを無制限に繰り返します）。
- 複数回の測定が設定されている場合は、プログラムされているトリガ時間間隔の終了を待って次回の測定を開始します。

図 4-2
測定トリガシーケンス



前面パネルトリガ

前面パネルからトリガ発生をコントロールするには、まず CONFIG を押してから TRIG を押し、続いて以下の方法でトリガパラメータを設定します。

TRIGGER-IN — 以下のオプションを選択してトリガ発生源を指定します。

- IMMEDIATE: トリガが即時発生し、測定値の取り込みが可能になると直ちにユニットがスタートします（例：ソース出力がオンになった直後）。
- MANUAL: 装置に対して読み込み開始トリガを発生するには前面パネルの TRIG キーを押さなければなりません。

COUNT — トリガカウント（測定回数）を次のように設定します。

- FINITE: プログラムされたトリガカウント（1 ～ 99999）に達するまで測定サイクルを繰り返します。
- INFINITE: 強制的に停止されるまで測定サイクルを無限に反復します。

INTERVAL — 測定から次回の測定までの時間間隔（0 ～ 999.999 秒）を指定します（COUNT が 1 以上の値に設定されている場合）。

DELAY — トリガ発生から測定開始までのディレイ（0 ～ 999.999 秒）を設定します。

前面パネルからトリガを発生させる例

次のようなセットアップを想定してトリガ発生の手順を説明します。

- 手動トリガ（TRIG キー使用）
- トリガカウント数無限（測定サイクルを無限に繰り返します）
- 測定から次回の測定までの時間間隔：1 秒
- ディレイ（トリガ発生から測定開始）：2 秒

この例では次のようにセットアップします。

1. まず CONFIG、続いて TRIG を押します。
2. TRIGGER-IN を選択し、ENTER または回転ノブを押します。続いて MANUAL を選択してから ENTER または回転ノブを押します。
3. COUNT を指定して INFINITE を選択し、ENTER または回転ノブを押します。
4. INTERVAL を選択してその値を 1 秒に設定し、ENTER または回転ノブを押します。
5. DELAY を選択してその値を 2 秒に設定し、ENTER を押します。
6. EXIT を押して通常の表示に復帰します。
7. OUTPUT をオンにしてから TRIG を押します。2 秒間のディレイが終了すると最初の測定が実行され、それ以後は 1 秒間隔で無限に測定サイクルを繰り返します。
8. 値の読み取りを終了させるには OUTPUT をオフにしてください。

リモートトリガ

リモートトリガ コマンド

表 4-6 にトリガコマンド一覧を示します。さらに詳しい内容についてはセクション 12 をご覧ください。

トリガイベント検出器は最後の trigger.wait 呼び出し以後にイベントが検出されたか否かを記憶しています。trigger.clear() コマンドはそのトリガのイベント検出器をクリアしてコマンドインターフェイストリガイベントのそれ以前の履歴を消去します。

trigger.wait 関数は、アクティブなコマンドインターフェイスとして GPIB が選択されている場合は timeout（秒）だけ GPIB GET コマンド（セクション 11 参照）または *TRG メッセージ（付録 C 参照）を待ち、それ以外のコマンドインターフェイスでは timeout（秒）だけ *TRG メッセージの到着を待ちます。これらのトリガイベントが少なくとも 1 つはすでに到着していたことを検出した場合、trigger.wait 関数は即時リターンします。trigger.wait 関数で trigger 秒だけの時間待ちをした後、イベント検出器は自動的にリセットされて次回の準備を行います（検出されたイベントの数には関係しません）。

表 4-6
Trigger コマンド

コマンド	説明
trigger.clear() triggered = trigger.wait(timeout)	現在ペンディングになっているトリガをクリア。 GET または *TRG トリガコマンドのタイムアウト時間終了を待ちます。

リモートトリガの例

次に示す例では、まずトリガをクリアし、タイムアウトを 30 秒にセットして GET/*TRG トリガ コマンドを待ちます。

```
smua.reset()                              --260x 型をデフォルト設定にリセット。
triggered = trigger.wait(30)              --GET/*TRG を待つ時間を 30 秒に設定。
smua.source.output =smua.OUTPUT_ON        --出力をオンにします。
*TRG                                      --読み込みトリガ発生
reading = smua.measure.v()                --電圧値読み取り。
```

基本的なソースメジャー手順

前面パネルからソースメジャー手順を実行

以下に説明する手順に従って 260x 型ソースメータのソースメジャー操作を行ってください。ソースメータと DUT はセクション 3 に説明した方法ですでに接続されているものとします。

警告 **出力端子とガード端子には危険な高電圧が印加される可能性があります。感電事故防止のため、出力がオンの状態では絶対に 260x 型の接続や接続の切り離し作業を行わないでください。場合によっては致命的な傷害を引き起こす可能性があります。出力端子に接続されているケーブルを操作するときは、必ず事前に前面パネルから電源を切るか、またはソースメータ背面の電源ケーブルを引抜いてください。装置をスタンバイ状態に入れたとしても、ハードウェアやソフトウェアの障害によって出力が完全にオフになっていない可能性があります。**

ステップ 1：ソースレベルの選択と設定

ソースを選択してその値を設定するために以下の操作を行います。

1. SRC を押して V-Source または I-Source を選択します（ディスプレイのソース表示フィールドに対応する単位が表示されます）。点滅表示されている桁（カーソル）が現在変更の対象となっている個所を示します。
2. 変更したい桁の位置へカーソルを移動させて回転ノブを押すことにより、EDIT（編集）モードへ入ります（EDIT ランプ点灯）。
3. RANGE キーを使用して設定したい値を含むレンジを選択します（レンジの詳細についてはセクション 6 をご覧ください）。確度を高めるためには使用可能な最低のソースレンジを選択してください。
4. 希望するソースレベルの値を入力し、最後に ENTER または回転ノブを押して変更作業を終了します。

ステップ 2：コンプライアンスリミットの設定

以下の手順に従ってコンプライアンスリミットの値を設定します。

1. 2602 型のシングルチャンネル表示モードまたは 2601 型の場合は LIMIT キーを押してください。
2. 2602 型デュアルチャンネル表示モードの場合は、まず CONFIG を押してから LIMIT キーを押し、続いて CURRENT または VOLTAGE のいずれかを選択します。最後に ENTER キーまたは回転ノブを押してください。
3. 変更したい桁の位置へカーソルを移動させて回転ノブを押すことにより EDIT（編集）モードへ入ります（EDIT ランプ点灯）。
4. 希望するリミット値を入力し、最後に ENTER または回転ノブを押して変更作業を終了します。

ステップ 3 : 測定機能とレンジの選択

以下の手順に従って測定機能とレンジを選択します。

1. 2602 型をシングルチャンネル表示モードへ入れ、次に MEAS または MODE を押して希望する測定機能を選択します。
2. RANGE キーを使用して希望する測定レンジを選択するか、または AUTO RANGE キーで自動レンジ機能をアクティブにします。この選択にあたっては以下の点に注意が必要です。
   - ソースを測定する場合（具体的には Source V Measure V）には RANGE キーで測定レンジを選択することはできません。ソースレンジを選択することで自動的に測定レンジが決まります。
   - ソース自体を測定するのではない場合（例：Source V Measure I）は手動で測定レンジを選択するか、または自動レンジ機能を使用します。手動でレンジを選択する場合は、測定確度の観点から使用可能な最も低いレンジを使用してください。自動レンジを選択した場合はソースメータが自動的に最も高い感度の得られるレンジへ移行します。

ステップ 4 : 出力オン

ON/OFF OUTPUT キーを押して出力をオンにします。出力がオンになると OUTPUT ランプが点灯します。

ステップ 5 : ディスプレイ上の指示値を観察

ディスプレイ上の指示値を観察します。必要ならば TRIG キーを押してトリガを発生させ、装置に新たに読み取りを行わせてください。シングルチャンネル表示モードの場合は読み取り値が一番上の行に表示され、ソースとリミット値が一番下の行に表示されます。

ステップ 6 : 出力オフ

出力を観察した後は ON/OFF OUTPUT キーを押して出力をオフにしてください。出力がオフになると OUTPUT ランプも消灯します。

ソースメジャー手順のリモート操作

基本的なソース メジャー操作をリモート実行することができます。そのためには正しいコマンドを正しい順序で送らなければなりません。以下の文では簡単なプログラミング例を用いてリモート操作のための基本的なコマンドについて説明します。

基本的なソースメジャー コマンド

基本的なソースメジャーコマンドをまとめて表 4-7 に示します。これらのコマンドの使用法についてさらに詳しくはセクション 12 をご覧ください。

表 4-7
基本的なソースメジャー コマンド

コマンド[1]	説明
smuX.measure.autorangei = smuX.AUTORANGE_ON	電流レンジを自動設定
smuX.measure.autorangev = smuX.AUTORANGE_ON	電圧レンジを自動設定
smuX.measure.autorangei = smuX.AUTORANGE_OFF	電流レンジの自動設定を解除
smuX.measure.autorangev = smuX.AUTORANGE_OFF	電圧レンジの自動設定を解除
smuX.measure.rangei = rangeval	電流測定レンジを設定
smuX.measure.rangev = rangeval	電圧測定レンジを設定
reading = smuX.measure.i()	電流値読み取り要求
reading = smuX.measure.v()	電圧値読み取り要求
reading = smuX.measure.iv()	電流と電圧値の読み取り要求
reading = smuX.measure.r()	抵抗値読み取り要求
reading = smuX.measure.p()	電力値読み取り値要求
smuX.source.autorangei = smuX.AUTORANGE_ON	電流ソースレンジを自動設定
smuX.source.autorangev = smuX.AUTORANGE_ON	電圧ソースレンジを自動設定
smuX.source.autorangei = smuX.AUTORANGE_OFF	電流ソースレンジの自動設定を解除
smuX.source.autorangev = smuX.AUTORANGE_OFF	電圧ソースレンジの自動設定を解除
smuX.source.func = smuX.OUTPUT_DCVOLTS	電圧ソース機能を選択
smuX.source.func = smuX.OUTPUT_DCAMPS	電流ソース機能を選択
smuX.source.leveli = sourceval	電流ソース値を設定
smuX.source.levelv = sourceval	電圧ソース値を設定
smuX.source.limiti = level	電流リミットを設定
smuX.source.limitv = level	電圧リミットを設定
smuX.source.output = smuX.OUTPUT_ON	ソース出力 ON
smuX.source.output = smuX.OUTPUT_OFF	ソース出力 OFF
smuX.source.rangei = rangeval	電流ソースのレンジ設定
smuX.source.rangev = rangeval	電圧ソースのレンジ設定
smuX.sense = smuX.SENSE_LOCAL	ローカルセンシング（2 線式）
smuX.sense = smuX.SENSE_REMOTE	リモートセンシング（4 線式）

1. 2601 型：smuX = smua、2602 型：smuX = smua (Channel A) または smub (Channel B)

データ出力要求

データ出力を要求するには適切な測定コマンドを引数として与えて print コマンドを実行します。例えば、次のコマンドはチャンネル A に対して電流値のデータ出力を要求します。

```
print(smua.measure.i())
```

ソースメジャープログラミングの例

基本的なソースメジャー操作を実行するコマンドシーケンス例を下に示します。これらのコマンドにより、ソースメータは次のように設定されます。

- ソース機能とレンジ：電圧、6V
- ソースの出力レベル：5V
- 電流コンプライアンス：10mA
- メジャー機能とレンジ：電流、10mA

```
smua.reset()                                --260x 型のデフォルト設定呼び戻し
smua.source.func = smua.OUTPUT_DCVOLTS--電圧ソース機能を選択
smua.source.rangev = 6                      --ソースレンジ 6V に設定
smua.source.levelv = 5                      --電圧ソースを 5V に設定
smua.source.limiti = 10e-3                  --電流リミットを 10mA に設定
smua.measure.rangei = 10e-3                 --電流レンジを 10mA に設定
smua.source.output =smua.OUTPUT_ON          --出力 ON
print(smua.measure.i())                     --電流値のデータ出力値を要求
smua.source.output =smua.OUTPUT_OFF         --出力 OFF
```

測定（メジャー）専用

標準的なソースメジャー動作を行わせることに加えて、ソースメータを電圧または電流の測定のみに使用することも可能です。ソースメータを電流 / 電圧の測定専用として使用するには次のステップを実行します。

1. ソースメジャー機能を選択します。

   電圧測定用として（電圧計）— SRC を押して I-Source を選択し、続いて MEAS を押して電圧測定機能を選択します。

   電流測定用として（アンメータ）— SRC を押して V-Source を選択し、続いて MEAS を押して電流測定機能を選択します。

2. ソースとコンプライアンスのレベルを設定します。

   「前面パネルからソースメジャー手順を実行」（p.4-13）のステップ 1、2 で説明した変更手順を使用して、ソースとコンプライアンスのレベルを次のように編集します。

a. 最も低いソースレンジを選択してソースレベルをゼロ（000.000nA または 000.000mV）に設定します。
b. コンプライアンスレベルを予想される測定値よりも高いレベルに設定します。

注意 **ソースメータをボルトメータ専用として使用するときは、電圧コンプライアンスを必ず被測定電圧よりも高く設定しなければなりません。電圧コンプライアンス値が測定値よりも低くなると過剰な電流（<150mA）がソースメータに流れ込み、正しい測定を行うことができなくなります。**

3. レンジを選択します。

   RANGE キーを使用して固定された測定レンジを選択します。予期される読み取り値はすべてこのレンジに収まる必要があります。確度を高めるためには適用可能な最小レンジを使用してください。

   ソース機能とは反対の機能を測定する場合は、固定の代わりに AUTO レンジを使用することができます。その場合、ソースメータは自動的に最も感度の高いレンジに移行します。

4. 測定対象の電圧または電流を接続します。2 線式接続（セクション 3 参照）を使用して DUT をソースメータに接続します。
5. ON/OFF キーを押して出力を ON にします。
6. ディスプレイの指示値を読み取ります（必要ならば TRIG キーを押してください）。読み取りが終了したならば出力を OFF に戻してください。

シンク操作

シンクとして動作しているソースメータ（V と I が逆極性）は、電力を供給するのではなく電力を消費します。外部のソース（例えばバッテリ）やエネルギー蓄積デバイス（例えばキャパシタ）によって動作が強制的にシンク領域に移行する場合があります。

例えば、+10V にプログラムされている V-Source を 12V のバッテリに接続（In/Out HI をバッテリの高電圧側へ接続）すると、第 2 象限（Source +V のときに電流測定）においてシンク動作が起こります。

注意 **I-Source をシンクとして使用するときは常に V-Compliance を外部に接続した電圧レベルよりも高いレベルに設定してください。この条件を満たさない条件で使用すると過剰な電流（102mA）がソースメータに流れ込み、正しい測定を行うことができなくなります。詳しくは「コンプライアンスリミット」（p.4-3）の説明をご覧ください。**

シンク動作時に守るべきリミット値が「動作限界条件」（p.8-7）に示されています。

抵抗測定

抵抗値の計算

抵抗として表示される値はソース電流と実測された電圧から次式に従って計算されます。

$$R = V/I$$

ここに： R は抵抗計算値、
V は実測された電圧、
I はソース電流の大きさです。

抵抗測定のレンジ設定

前面パネルの抵抗測定機能はレンジ切換えを使用しません。V/I 計算値をディスプレイの表示形式に自動的に合わせて表示してくれます。抵抗指示値が非常に小さい（<1mΩ）場合は数字の先頭部に “0” が付加されます。

基本的な抵抗測定操作

抵抗測定では以下のステップを実行します。以下に説明する操作は、ソースメータと DUT がセクション 3 に説明した方法ですでに接続されていることを仮定しています。

警告 **出力端子とガード端子には危険な高電圧が印加される可能性があります。感電事故防止のため、出力がオンの状態では絶対に 260x 型の接続や接続の切り離し作業を行わないでください。場合によっては致命的な傷害を引き起こす可能性があります。出力端子に接続されているケーブルを操作するときは、必ず事前に前面パネルから電源を切るか、またはソースメータ背面の電源ケーブルを引抜いてください。装置をスタンバイ状態に入れたとしても、ハードウェアやソフトウェアの障害によって出力が完全にオフになっていない可能性があります。**

1. 2602 型を使用する場合は DISPLAY キーを押してシングルチャンネル表示モードを選択してください。
2. SRC を押して電流ソース機能を選択し、続いて出力電流を予想される抵抗値に合わせて設定します。具体的な操作は「前面パネルからソースメジャー手順を実行」（p.4-13）のステップ 1 を参照してください。
3. LIMIT キーを押し、電圧リミット値を抵抗の両端に発生すると予想される電圧値よりも充分に高い値に設定します（抵抗値とプログラムされているソース電流値の両方を勘案してください）。具体的な操作は「前面パネルからソースメジャー手順を実行」（p.4-13）のステップ 2 を参照してください。
4. MEAS または MODE キーを押して電圧を表示させ、測定レンジとして AUTO が選択されていることを確認します。

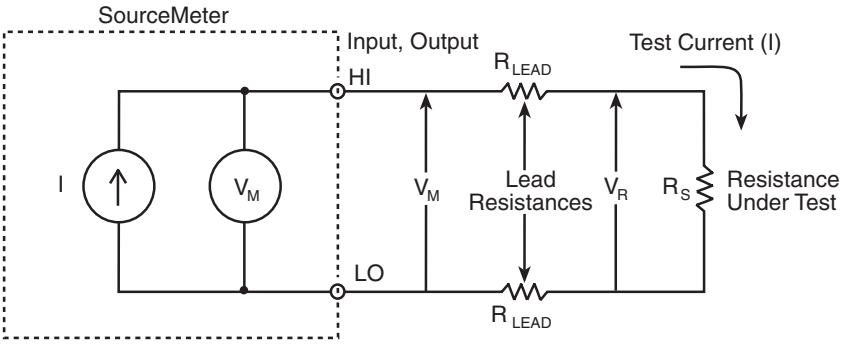
5. MEAS または MODE キーを押して抵抗値を表示させます。
6. 出力を ON にしてディスプレイに表示される値を読み取ります。連続的に読み取りを行いたい場合は TRIG キーを押してください。測定終了後は出力をオフにしておきます。

抵抗センシング

抵抗測定にも 2 線式と 4 線式の両方を使用することができます。（接続とセンシングモードの詳細についてはセクション 3 をご覧ください。）

2 線式には必要となる試験導線が 2 本だけであるという利点があります。しかし、図 4-3 に示すように、2 線式抵抗測定では試験導線自体の持つ抵抗が測定確度に大きな影響を与えることがあり、特に低い抵抗値の測定ではこの効果が顕著になります。これに対して図 4-4 に示す 4 線式センシングは 2 番目の試験導線セットを使用して測定対象両端の電圧降下を測定しますから、試験導線の持つ抵抗の影響を最小化もしくは完全に除去することができます。ソースメータは非常に高い入力インピーダンスを持ちますから、検出用導線に流れる電流は無視できるほど小さく、したがって測定される電圧はほぼ完全に被試験抵抗両端の電圧を反映した値になります。

図 4-3
2 線式抵抗センシング

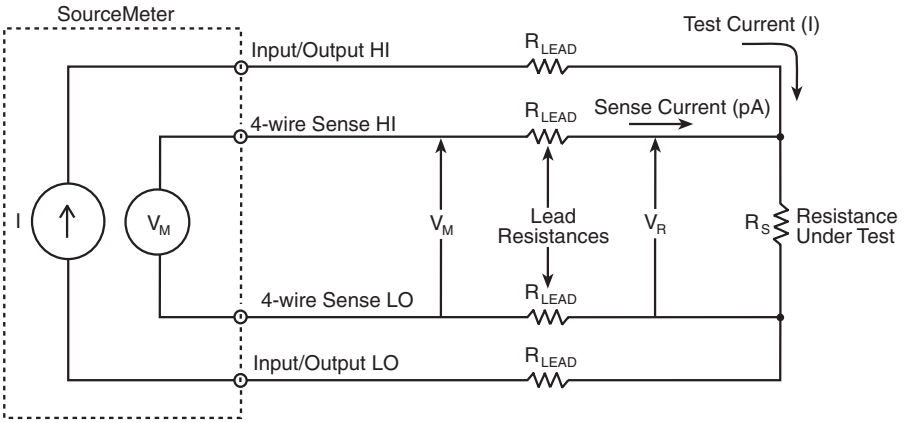


I = Current sourced by SourceMeter
V_M = Voltage measured by SourceMeter
V_R = Voltage across resistor

$$\text{Measured resistance} = \frac{V_M}{I} = R_S + (2 \times R_{LEAD})$$

$$\text{Actual resistance} = \frac{V_R}{I} = R_S$$

図 4-4
4 線式抵抗センシング



I = Current sourced by SourceMeter
V_M = Voltage measured by SourceMeter
V_R = Voltage across resistor

Because sense current is negligible, $V_M = V_R$

and measured resistance = $\frac{V_M}{I} = \frac{V_R}{I} = R_S$

センシング方法の選択

前面パネルからセンスモードを選択

センシングモードの選択方法：

1. まず CONFIG キー、続いて MEAS キーを押します。V-MEAS を選択し、ENTER または回転ノブを押します。
2. SENSE-MODE を選択して ENTER を押します。
3. 2-WIRE または 4-WIRE のいずれか希望する方を選択し、ENTER または回転ノブを押します。

センスモードのリモート選択

センスモードのリモート選択には smuX.sense コマンドを使用します。例えば、4 線式センシングをアクティブにするには次のコマンドを送ります。

```
smua.sense = smua.SENSE_REMOTE
```

詳細については表 4-7（p.4-15）およびセクション 12 をご覧ください。

抵抗測定のリモートプログラミング

以下の項では典型的な抵抗測定のプログラム例を示しながら、抵抗測定をリモートプログラミングするために必要な基本コマンドの要点をまとめて説明します。

抵抗測定のリモートコマンド

抵抗値を読取るには次のコマンドを使用します。

```
reading = smuX.measure.r()
```

ソースと測定機能のセットアップに必要となる他のコマンドが表 4-7（p.4-15）にまとめられています。これらのコマンドの更に詳しい説明についてはセクション 12 をご覧ください。

抵抗測定のプログラミング例

典型的な抵抗測定に使用できるコマンドシーケンスを次に示します。これらのコマンドはソースメータを次のように設定します。

- ソース機能：電流、10mA レンジ、10mA 出力
- 電圧測定レンジ：自動
- 電圧コンプライアンス：10V
- センスモード：4 線式

```
smua.reset()                                    --260x 型のデフォルト設定呼び戻し
smua.source.func = smua.OUTPUT_DCAMPS           --電流ソース機能を選択
smua.source.rangei = 10e-3                      --ソースレンジを 10mA に設定
smua.source.leveli = 10e-3                      --電流ソースを 10mA に設定
smua.source.limitv = 10                         --電圧リミットを 10V に設定
smua.sense = smua.SENSE_REMOTE                  --4 線式抵抗測定をアクティブ化
smua.measure.autorangev = smua.AUTORANGE_ON     --電圧レンジを自動に設定
smua.source.output = smua.OUTPUT_ON             --出力 ON
print(smua.measure.r())                         --抵抗値読み取り
smua.source.output = smua.OUTPUT_OFF            --出力 OFF
```

電力測定

電力の演算

電力として表示される値はソースの値と実測された電流または電圧値から次式に従って演算されます。

$$P = V \times I$$

ここに： P は電力の計算値、
V はソース電圧、または実測された電圧、
I は実測電流、またはソース電流を表します。

電力測定の基本手順

電力測定では以下のステップを実行します。以下に説明する操作は、ソースメータと DUT がセクション 3 に説明した方法ですでに接続されているものと仮定しています。

警告 **出力端子とガード端子には危険な高電圧が印加される可能性があります。感電事故防止のため、出力がオンの状態では絶対に 260x 型の接続や接続の切り離し作業を行わないでください。場合によっては致命的な傷害を引き起こす可能性があります。出力端子に接続されているケーブルを操作するときは、必ず事前に前面パネルから電源を切るか、またはソースメータ背面の電源ケーブルを引抜いてください。装置をスタンバイ状態に入れたとしても、ハードウェアやソフトウェアの障害によって出力が完全にオフになっていない可能性があります。**

1. 2602 型を使用する場合は DISPLAY キーを押してシングルチャンネル表示モードを選択してください。
2. ソース機能と値を設定します。まず SRC を押して必要なソース機能（電圧または電流）を選択し、その出力値を希望の値にセットします。具体的な操作については「前面パネルからソースメジャー手順を実行」（p.4-13）のステップ 1 を参照してください。
3. LIMIT キーを押し、続いて測定対象となる DUT にかかると予期される電圧 / 電流を考慮して充分に高いリミット値を設定します。具体的な操作については「前面パネルからソースメジャー手順を実行」（p.4-13）のステップ 2 を参照してください。
4. MEAS または MODE キーを押して電力値を表示させます。
5. 出力を ON にしてディスプレイに表示される値を読み取ります。連続的に読み取りを行いたい場合は TRIG キーを押してください。
6. 測定終了後は出力を OFF にしておきます。

電力測定のリモートプログラミング

以下の項では典型的な電力測定のプログラムを示しながら、電力測定のリモートプログラミングに必要な基本コマンドをまとめて説明します。

リモート電力コマンド

電力値を読み取るには次のコマンドを使用します。

```
reading = smuX.measure.p()
```

ソースと測定機能のセットアップに必要となる他のコマンドが表 4-7（p.4-15）にまとめられています。これらのコマンドの更に詳しい説明についてはセクション 12 をご覧ください。

電力プログラミングの例

典型的な電力測定に使用できるコマンドシーケンスを次に示します。これらのコマンドはソースメータを次のように設定します。

- ソース機能：電圧、6V レンジ、5V 出力
- 電流測定機能とレンジ：電流、自動
- 電流コンプライアンス：50mA

```
smua.reset()                                    --260x 型のデフォルト設定呼び戻し
smua.source.func = smua.OUTPUT_DCVOLTS          --電圧ソース機能を選択
smua.source.rangev = 6                          --ソースレンジを 6V に設定
smua.source.levelv = 5                          --電圧ソースを 5V に設定
smua.source.limiti = 50e-3                      --電流リミットを 50mA に設定
smua.measure.autorangei = smua.AUTORANGE_ON--電流レンジを自動に設定
smua.source.output = smua.OUTPUT_ON             --出力 ON
print(smua.measure.p())                         --電力値読み取り
smua.source.output = smua.OUTPUT_OFF            --出力 OFF
```

5
スイープ操作

セクション 5 のトピックス

概要 , 5-2
- このセクションのまとめ , 5-2
- スイープのまとめ , 5-2

スイープ特性 , 5-3
- 線形階段スイープ , 5-3
- 対数階段スイープ , 5-4
- パルススイープ , 5-6
- カスタム（リスト）スイープ , 5-7
- スイープ測定値の保存 , 5-8

スイープ関数 , 5-8
- 階段スイープ関数 , 5-9
- パルススイープ関数 , 5-10
- カスタムスイープ関数 , 5-10

スイープ実行 , 5-11
- 前面パネル , 5-11
- スイーププログラミングの例 , 5-11

概要

このセクションのまとめ

「スイープの概要」でスイープの各種タイプ（線形階段、対数階段、カスタム、パルス）を簡単に説明し、このセクションのそれ以後の項ではそれぞれのスイープモードの特性、コマンド、プログラミングについて次の順で詳しく説明します。

- 「スイープのまとめ」、p.5-2
- 「スイープ特性」、p.5-3
- 「スイープ関数」、p.5-8
- 「スイープ実行」、p.5-11

スイープのまとめ

図 5-1 に示すように、260x 型ソースメータをファクトリスクリプトで動作させて、何種類かのスイープパターンを発生させることができます。

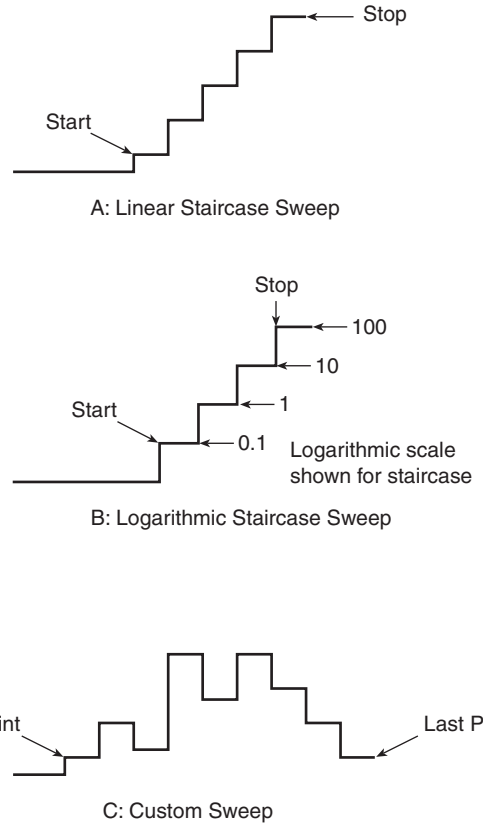
線形階段スイープ – このタイプのスイープでは電圧または電流が特定のステップにしたがって上昇または下降してゆきます（スタート電流で始まり、ストップ電流で終わります）。上昇方向へ変化する線形階段スイープの例を図 5-1A に示します。

対数階段スイープ – このケースでは電流または電圧が対数的に増加または減少し、スタート電圧 / 電流から始まってストップ電圧 / 電流で終了します。上昇方向へ変化する対数階段スイープの例を図 5-1B に示します。

カスタム（リスト）スイープ – カスタムスイープでは 260x 型の出力電圧と電流レンジの範囲内でユーザが任意のステップをプログラムすることができます。典型的なカスタムスイープ（任意ステップ）の例を図 5-1C に示します。

パルススイープ – パルススイープでは電流を固定するパルスと電圧を固定するパルスの 2 通りのスイープが可能です。これらのパルススイープもこれまでに説明したスイープモードと良く似ていますが、上記のスイープモードのように幅が一定ではなく各ポイントで特定のパルス幅を指定してパルスを発生させる点が異なります。

図 5-1
階段スイープのタイプ



スイープ特性

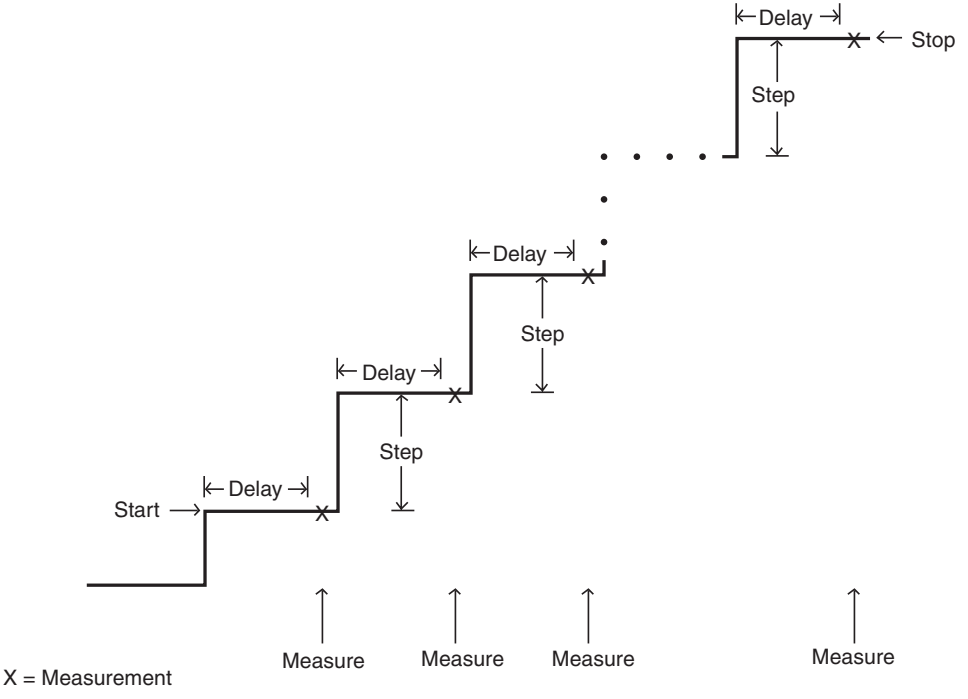
線形階段スイープ

図 5-2 に示すように、このタイプのスイープはスタート電圧 / 電流から始まってストップ値まで変化し、各ステップごとにディレイの終了を待って測定が行われます。このスイープモードではソース機能、チャンネル、スタートとストップレベル、スイープポイント数、ディレイ（各ステップごとのソーシング開始から測定までの時間）等のパラメータをプログラムすることができます。ステップのサイズはスタートとストップレベル、およびスイープポイント数から次式によって決定されます。

step = (stop - start) / (points - 1)

スイープの方向はスタートとストップパラメータ値の相対的な関係によって上昇と下降方向への変化いずれも可能です。スイープ開始とともに出力がスタートソースレベルへ変化し、それ以後ストップレベルに達するまで同一ステップレベルで変化してゆきます。ディレイパラメータは各スイープステップでの測定開始までの時間を指定します。

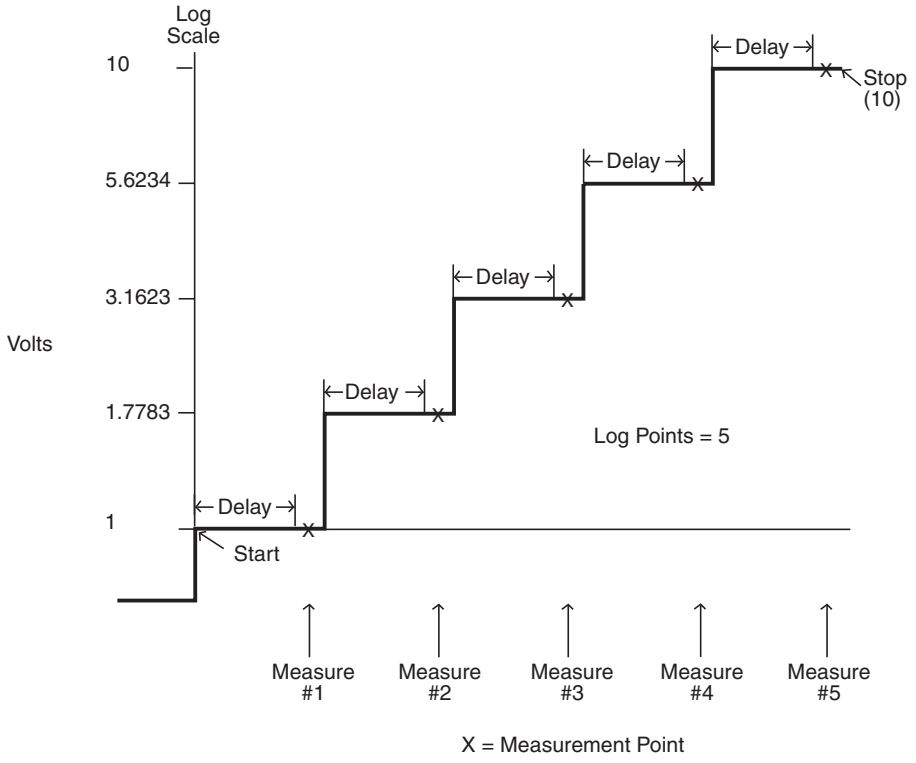
図 5-2
線形階段スイープ



対数階段スイープ

このスイープモードも線形階段スイープとよく似ていますが、スイープの縦軸が図 5-3 に示すように対数スケールで変化します。この図に示すのは 1V から 10V までを 5 ポイントで対数スイープする例です。スイープの方向はスタートとストップパラメータ値の相対的な大小関係によって上昇と下降方向への変化いずれも可能です。

図 5-3
対数階段スイープ（1V から 10V、5 ステップ）



対数スイープではソース機能、チャンネル、スタートとストップレベル、ディレイ、およびスイープが含む測定ポイント数などのパラメータをプログラムすることができます。スタート、ストップ、およびポイント数パラメータを指定することによってスイープの対数ステップサイズが決まります。ディレイパラメータは各スイープステップでの測定開始までの時間を指定します。

図 5-3 に示すスイープのステップサイズは次式によって計算されます。

$$\text{Log Step Size} = \frac{\log10(\text{stop}) - \log10(\text{start})}{\text{Points} - 1}$$

$$= \frac{\log10(10) - \log10(1)}{5 - 1}$$

$$= \frac{(1 - 0)}{4}$$

$$= 0.25$$

この式から、このスイープの 5 段階の対数ステップはそれぞれ 0、0.25、0.50、0.75、および 1.00 となります。これらのポイントにおける実際のソースメータ出力値を表 5-1 に示します（電圧レベルは対数ステップの逆対数をとった値です）。

表 5-1
対数スイープポイント

測定ポイント	対数ステップ	ソースレベル（V）
ポイント 1	0	1
ポイント 2	0.25	1.7783
ポイント 3	0.50	3.1623
ポイント 4	0.75	5.6234
ポイント 5	1.0	10

このスイープがスタートすると、出力がまずスタートレベル（1V）へ変化し、続いて対称な対数ポイントを順次スイープしてゆきます。各ステップへ移行してから測定開始までの時間は測定ディレイによって決まります。

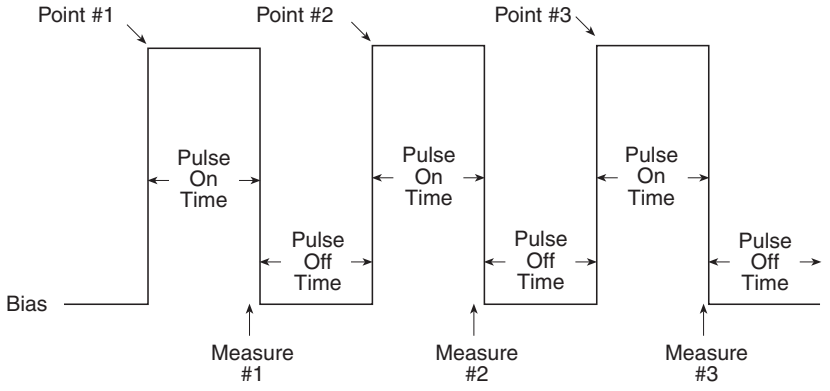
パルススイープ

固定パルススイープは電圧または電流の固定されたパルスを発生します。パルススイープではソース機能、バイアルレベル、パルスレベル、パルス数、パルスのオン時間とオフ時間、およびソースメータのチャンネル（A または B）などのパラメータをプログラムすることができます。

スイープが始まると、出力はバイアスレベルから最初のパルスの “ON” レベルへと変化します。各レベルでのパルス持続時間は ON 時間をプログラムすることにより、パルス終了から次のパルスまでの時間は OFF 時間をプログラムすることによって決定されます。

3 個のパルスだけを含む単純なパルススイープの例を図 5-4 に示します。ON 時間の設定によってパルス幅が決まり、OFF 時間の設定によってパルス間の時間が決まることに注意してください。パルスレベルはすべてのパルスに共通です。

図 5-4
パルススイープの例



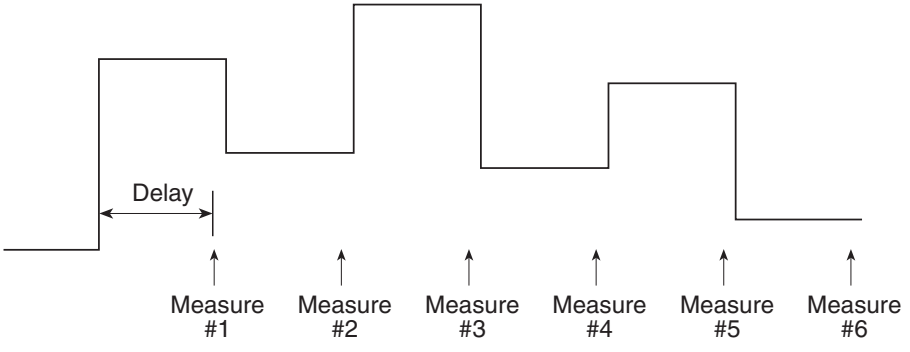
カスタム（リスト）スイープ

カスタムスイープでは任意のステップを含むスイープをユーザが設計することができます。カスタムスイープでプログラム可能なパラメータにはソース機能、チャンネル、スイープ値リスト、ディレイおよびポイント数などが含まれます。

スイープが始まると、まず出力がスイープリストの最初の値へ変化し、その後はプログラムされたステップの順序に従ってステップが変化してゆき、最後のステップ終了によってスイープが終わります。各ステップへ移行してから測定開始までの時間はディレイによって決まります。

6 個の測定ポイントを持つカスタムスイープの例を図 5-5 に示します。スイープ開始とともにスイープリストの最初のポイントが出力され、それ以後はプログラムされた順序に従って順次スイープポイントをスキャンしてゆきます。

図 5-5
カスタムスイープの例



スイープ測定値の保存

スイープを実行すると測定値は自動的に不揮発性メモリ（Buffer 1）に書き込まれ、後で自由に呼び出すことができます。スイープデータを呼び出す方法は次の通りです。

- **前面パネル**：まず RECALL キーを押し、続いてチャンネルと Buffer 1 を選択してから最後に回転ノブまたはカーソルキーを使用して読み取り値の数を指定します。
- **リモート**：printbuffer コマンドを使用して smua.nvbuffer1 (channel A) または smub.nvbuffer1 (channel B) からのデータ読み出しを要求します。

バッファからデータを読み出す方法の詳細についてはセクション 7 をご覧ください。

スイープ関数

階段 / パルス / カスタムスイープを実行するために使用する関数を以下に説明します。ファクトリスクリプトを使用してスイープを実行する方法の詳細についてはセクション 13 をご覧ください。

> **注** ここで説明する関数はこの資料作成の時点における最新のファクトリスクリプトを示しています。スイープに使用できるファクトリスクリプトはそれ以後も更新されている可能性があります。最新情報についてはリリースノートをご覧ください。
>
> その他、様々な試験に利用できる各種のユーザスクリプトが用意されています。是非弊社 Web サイト（www.keithley.com）をご覧ください。

階段スイープ関数

線形階段スイープ、および対数階段スイープで使用する関数を表 5-2 に示します。

表 5-2
階段スイープ関数

コマンド	説明
SweepILinMeasureV(smu, starti, stopi, stime, points) smu start i stopi stime points	リニアな電流ソーススイープを定義： Smu:smua はチャンネル A、smub はチャンネル B を表す スタート電流の値（A） ストップ電流の値（A） 整定時間（ソースメジャーディレイ：秒） ポイント数 (≥2)
SweepVLinMeasureI(smu, startv, stopv, stime, points) smu startv stopv stime points	リニアな電圧ソーススイープを定義： Smu:smua はチャンネル A、smub はチャンネル B を表す。 スタート電圧値（V） ストップ電圧値（V） 整定時間（ソースメジャーディレイ：秒） ポイント数 (≥2)
SweepILogMeasureV(smu, starti, stopi, stime, points) smu starti stopi stime points	対数電流ソーススイープを定義： Smu:smua はチャンネル A、smub はチャンネル B を表す。 スタート電流の値（A） ストップ電流の値（A） 整定時間（ソースメジャーディレイ：秒） ポイント数 (≥2)
SweepVLogMeasureI(smu, startv, stopv, stime, points) smu startv stopv stime points	対数電圧ソーススイープを定義： Smu:smua はチャンネル A、smub はチャンネル B を表す。 スタート電圧値（V） ストップ電圧値（V） 整定時間（ソースメジャーディレイ：秒） ポイント数 (≥2)

パルススイープ関数

パルススイープで使用する関数を表 5-3 に示します。

表 5-3
パルススイープ関数

関数	説明
PulseIMeasureV(smu, bias, level, ton, toff, points) smu bias level ton toff points	電流ソース固定パルススイープを定義： Smu:smua はチャンネル A、smub はチャンネル B を表す。 DC バイアス電流レベル（A） パルス ON 時のソース電流（A） パルスの ON 時間（秒） パルスの OFF 時間（秒） パルス測定サイクルの数
PulseVMeasureI(smu, bias, level, ton, toff, points) channel bias level ton toff points	電圧ソース固定パルススイープを定義： Smu:smua はチャンネル A、smub はチャンネル B を表す。 DC バイアス電圧レベル（V） パルス ON 時のソース電圧（V） パルスの ON 時間（秒） パルスの OFF 時間（秒） パルス測定サイクルの数

カスタムスイープ関数

カスタム（リスト）スイープで使用する関数を表 5-4 に示します。

表 5-4
カスタムスイープ関数

コマンド	説明
SweepIListMeasureV(smu, ilist, stime, points) smu ilist stime points	電流リストスイープを定義： Smu:smua はチャンネル A、smub はチャンネル B を表す。 電流値リスト（A） 整定時間（ソースメジャーディレイ：秒） ポイント数 (≥2)
SweepVListMeasureI(smu, vlist, stime, points) smu vlist stime points	電圧リストスイープを定義： Smu:smua はチャンネル A、smub はチャンネル B を表す。 電圧値リスト（V） 整定時間（ソースメジャーディレイ：秒） ポイント数 (≥2)

スイープ実行

前面パネル

スイープを実行するには、まず LOAD キーを押してから実行したい試験を選択します。続いて RUN キーを押してから表示されるプロンプトに従って試験の必要事項を入力します（スイープパラメータについては表 5-2 から表 5-4 を参照）。ファクトリスクリプトの使用法について詳しくはセクション 13 をご覧ください。

取得したスイープデータ（Buffer 1 に保存）へのアクセスには RECALL キーを使用します。バッファについて詳しくはセクション 7 をご覧ください。

スイーププログラミングの例

続くページでは 3 種類のスイープタイプそれぞれについてプログラミングを行いスイープを実行する方法を説明します。こららの手順それぞれが標準的なスイープ例で使用されるコマンドを含んでいます。これらの例で使用するパラメータを表 5-5 にまとめて示します。

注　これらのプログラム例は本資料作成時における最新の工場スイープスクリプトを利用しています。それ以降変更が発生しているかも知れません。詳しくはリリースノートをご覧ください。

表 5-5
スイープ例が使用するパラメータ

スイープタイプ	スイープ例のパラメータ
線形階段スイープ（p.5-12）	スタート電流： 1 mA ストップ電流：10mA ポイント数：10 整定時間：0.1 秒
パルス電流スイープ（p.5-13）	バイアス電流： 1 mA ON 電流：10mA パルス ON 時間：10ms パルス OFF 時間：50ms ポイント数：10
カスタム（リスト）スイープ（p.5-14）	5 ポイント：3V、1V、4V、5V、2V 整定時間 0.1 秒

線形階段スイープの例

1. **ソース機能の設定**

   例 – 以下のコマンドはデフォルト設定を呼び戻し、コンプライアンスを 1V に設定します。

```
smua.reset()                    --260x 型のデフォルト設定呼び戻し
smua.source.limitv = 1          --コンプライアンスを 1V にセット
```

2. **スイープモードの設定**

   例 – 以下のパラメータにより 1mA から 10mA を 10 ポイント（整定時間 0.1 秒）で電流スイープする線形階段スイープが設定されます。

```
SweepILinMeasureV(smua, 1e-3,   --線形階段スイープ、チャンネル A、
10e-3, 0.1, 10)                   1mA ～ 10mA、遅延 0.1 秒、10 ポイント
```

3. **スイープ実行**

```
smua.source.output =            --出力 ON
smua.OUTPUT_ON
waitcomplete()                  --スイープ完了待ち
```

4. **出力 OFF**

   スイープが完了したならば、このコマンドを使用してソース出力を OFF にしておきます。

```
smua.source.output = smua.OUTPUT_OFF
```

5. **読み取り値を要求**

   次の方法で Buffer 1 に保存されている読み取り値を要求します。

```
printbuffer(1, 10, smua.nvbuffer1.readings)
```

パルススイープの例

1. **ソース機能の設定**

   例 – 以下のコマンドはデフォルト設定を呼び戻し、コンプライアンスを 10V に設定します。

```
smua.reset()                    --260x 型のデフォルト設定呼び戻し
smua.source.limitv = 10         --コンプライアンスを 10V にセット
```

2. **スイープモードの設定**

   例 – 以下のパラメータにより 10mA 電流パルススイープが選択されます。パルスのオン時間 10ms、オフ時間は 50ms、10 パルスで 1 測定サイクルを構成します。

```
PulseIMeasureV(smua, 1e-3,      --パルス電流スイープ、チャンネル
10e-3, 10e-3, 50e-3, 10)          A、バイアス 1mA、レベル 10mA、
                                  パルスのオン時間 10ms、パルスオ
                                  フ時間 50ms、10 サイクル
```

3. **スイープ実行**

```
smua.source.output =            --出力 ON
smua.OUTPUT_ON
waitcomplete()                  --スイープ完了待ち
```

4. **出力 OFF**

   スイープが完了したならば、このコマンドを使用してソース出力を OFF にしておきます。

```
smua.source.output = smua.OUTPUT_OFF
```

5. **読み取り値を要求**

   次の方法で Buffer 1 に保存されている読み取り値を要求します。

```
printbuffer(1, 10, smua.nvbuffer1.readings)
```

カスタムスイープの例

1. **ソース機能の設定**

   例 – 以下のコマンドはデフォルト設定を呼び戻し、コンプライアンスを 10mA に設定します。

```
smua.reset()                    --260x 型のデフォルト設定呼び戻し
smua.source.limiti = 10e-3      --コンプライアンスを 10mA にセット
```

2. **スイープモードの設定**

   例 – 以下のパラメータは測定ポイントとして 3V、1V、4V、5V、2V を指定したリストスイープを設定します（整定時間は 0.1 秒）。

```
vlist = {5, 3, 1, 4, 5, 2}      --電圧リストを定義
SweepVListMeasureI(smua, vlist, --リストスイープ、チャンネル A、
0.1, 5)                          vlist でステップ指定（3V、1V、
                                 4V、5V、2V）、ディレイ 0.1 秒、
                                 5 ポイント
```

3. **スイープ実行**

```
smua.source.output =            --出力 ON
smua.OUTPUT_ON
waitcomplete()                  --スイープ完了待ち
```

4. **出力 OFF**

   スイープが完了したならば、このコマンドを使用してソース出力を OFF にしておきます。

```
smua.source.output = smua.OUTPUT_OFF
```

5. **読み取り値を要求**

   次の方法で Buffer 1 に保存されている読み取り値を要求します。

```
printbuffer(1, 5, smua.nvbuffer1.readings)
```

6
レンジ、桁数、速度、リラティブ、およびフィルタ

セクション 6 のトピックス

概要 , 6-2

レンジ , 6-2
- 選択可能なレンジ , 6-2
- ソースの最大値と読み取り値 , 6-3
- レンジ設定に関する制約事項 , 6-3
- 手動レンジ切換え , 6-3
- 自動レンジ切換え , 6-3
- 低域側のレンジリミット , 6-4
- レンジ設定についての注意事項 , 6-4
- レンジ設定のプログラミング , 6-5

桁数 , 6-7
- 表示分解能の設定 , 6-7
- 表示桁数のリモートプログラミング , 6-7

速度 , 6-8
- 速度設定 , 6-9
- 速度設定のリモートプログラミング , 6-9

リラティブ , 6-10
- 前面パネルからの リラティブ 操作 , 6-10
- リラティブ のリモートプログラミング , 6-11

フィルタ , 6-12
- フィルタタイプ , 6-12
- 応答時間についての注意事項 , 6-12
- 前面パネルからのフィルタコントロール , 6-12
- フィルタのリモートプログラミング , 6-15

概要

このセクションでは以下の機能の特性とスクリプトプログラミングの方法について詳しく説明します。

- 「レンジ」、p.6-2
- 「桁数」、p.6-7
- 「速度」、p.6-8
- 「リラティブ」、p.6-10
- 「フィルタ」、p.6-12

レンジ

測定確度はもちろん測定可能な最大信号も、どの測定レンジを選択するかによって影響を受けます。測定の途中でレンジ変更を含む切換操作を行うとディスプレイ上に点線が表示されることがあります（例：--.---- μA）。点線表示はその前の測定との間に少し時間が空いたことを示します。

選択可能なレンジ

260x 型ソースメータで使用可能なソースおよび測定レンジをまとめて表 6-1 に示します。

表 6-1
ソースと測定レンジ

電圧レンジ	電流レンジ
100mV	100nA
1V	1µA
6V	10µA
40V	100µA
	1mA
	10mA
	100mA
	1A
	3A

ソースの最大値と読み取り値

ある電圧または電流ソースレンジを選択すると、そのフルスケール出力はレンジ定格値の 101% となり、測定値のフルスケールはレンジ定格の 102% になります。例えば、ソースレンジとして 1V を選択したときのフルスケールソース値は ± 1.01V となり、測定レンジとして 100mA を指定した場合の読み取り値フルスケールは ± 102mA となります。入力のレベルがこれらの最大値を超えるとオーバーフローメッセージが表示されます。ただし、選択されているレンジの 100% に達すると装置が自動レンジ切換えを行います。

レンジ設定に関する制約事項

- V-Source レンジとして 40V を選択した場合の最大電流測定レンジは 1A です。I-Source レンジとして 3A を選択した場合の最大電圧測定レンジは 6V です。
- 電圧ソースで電流測定、または電流ソースで電圧測定を行う場合は、ソースと測定のレンジを独立して別々に設定することができます。ソースと測定が同じ種類である場合の測定レンジは、自動的にソースレンジにロックされます。

手動レンジ切換え

固定レンジの選択には RANGE 上矢印キーと下矢印キーを組み合わせて使用します。

- ソースレンジの設定：まず SRC を押し、続いて RANGE キーを使用してレンジを設定します。
- 測定レンジの設定：（2602 型の場合は最初にシングルチャンネル表示モードを選択）MEAS キーを押し、続いて RANGE キーを使用してレンジを設定します（電圧ソース - 電流測定、または電流ソース - 電圧測定）。

ある特定のレンジでオーバーフローメッセージが表示されるようであれば、レンジの範囲内の値が表示されるまで順次上位のレンジに切換えてください。最良の確度と分解能を得るには、オーバーフローが起こらない範囲で可能な最小のレンジを使用する必要があります。

自動レンジ機能がオフになっている場合は、レンジを超える値を持つ ICL コマンドをソースメータに送ることが可能であり、例えば 100mV レンジに 1V を送ることができます。ただしこの場合、装置は出力が ON になるまでエラーチェックを行いません。この状況が発生すると、ディスプレイには一連の疑問符シンボル（???.???）が表示されます。

自動レンジ切換え

ソースの自動レンジ切換えを使用するには、まず SRC を押してから AUTO RANGE を選択します。測定レンジの自動切換えを使用するときは、（2602 型の場合はシングルチャンネル表示モードを選択）MEAS を押してから AUTO RANGE を選択します。ソースまたは測定の自動レンジ切換えが選択されると AUTO ランプが点灯します。自動レンジ機能が選択されると、印加された信号に最適なソースレンジまたは測定レンジを装置が自動的に選択し、そのレンジの 100% に達すると自動的にレンジ切換えを行います。

ソース値の変更作業中はソース側の自動レンジ機能がオフになります。

低域側のレンジリミット

低域側のレンジにリミットを設定しておくと、260x 型の自動レンジ機能をオンにしたときに取りえるレンジの最小値がこのリミット値によって制限されます。この機能は、レンジ切換えが頻繁に起るような条件下で測定を行う場合の自動レンジ切換え整定時間を短くするために有効です。

レンジの低域リミットは以下の手順により Source V、Source I、Measure V、および Measure I に対して独立に設定が可能です。

1. CONFIG キーを押し、続いてソース（SRC）または測定（MEAS）を選択します。
2. 電圧 / 電流ソース、または測定項目を正しく選択してから ENTER または回転ノブを押します。
3. LOWRANGE を選択し、ENTER または回転ノブを押します。
4. 低域レンジを希望する値に設定し、ENTER または回転ノブを押します。
5. EXIT キーを使用してメニューから抜け出します。

レンジ設定についての注意事項

どのソース機能を選択するかにより、ソース側のレンジと測定側のレンジ選択に制限が生ずることがあります。また、出力の状態（ON/OFF）によってレンジ設定の方法が制約されることがあります。

ソース機能と測定機能が同じ種類（例えば電圧ソースで電圧測定）である場合、測定レンジは同じソースレンジにロックされます。しかし、その状態でソース機能を電流に切換えたとしても電圧測定レンジはその値を保ち、現在の電圧測定レンジがそのまま継続して使用されます。

例：

```
smua.source.func = smua.OUTPUT_DCVOLTS
smua.source.rangev = 1
smua.measure.rangev = 6
print(smua.measure.rangev)      -- ソースレンジにロックされていますから 1 を出力します。
smua.source.func = smua.OUTPUT_DCAMPS
print(smua.measure.rangev)      -- ユーザが指定したレンジである 6 を出力します。
```

ある機能に対して陽にソースまたは測定レンジを設定すると、その機能の自動レンジ切換えが解除されます。自動レンジ切換え機能は個々のソース、測定機能（電圧ソース、電流ソース、電圧測定、電流測定）ごとに独立してコントロールされます。デフォルト設定では、すべての機能の自動レンジ切換えがアクティブになっています。

出力 OFF の状態でレンジを切換えたとしてハードウェアの設定はその時点では更新されません。ただし、クエリコマンドに対しては出力が ON に変化したとき

に使用されるレンジ設定を返します。出力が ON の状態でレンジを設定した場合には新しい値が直ちに有効になります。

ソース側の自動レンジ切換えがアクティブである場合は、出力レベルがレンジをコントロールします。レベル設定後にレンジの問合せを行うと、装置がそのレベルに適当と判断したレンジ値が返されます。

自動レンジ機能がオフになっている場合は、レンジを超える値を持つ ICL コマンドであってもソースメータに送ることが可能であり、例えば 100mV レンジのときに 1V を送信することができます。装置は出力が ON になるまでエラーチェックを行ないませんから、この状況が発生するとディスプレイに一連の疑問符シンボル（???.???）を表示します。

測定側の自動レンジ切換えがオンのときに実際に切換えが起るのは、測定の取り込みが行われるときです。したがって、測定後にレンジを問い合わせることによって、その測定に実際に使われたレンジを取得することができます。

レンジ設定のプログラミング

レンジコマンド

測定 / ソースレンジをコントロールするために必要なコマンドをまとめて表 6-2 に示します。これらのコマンドについてさらに詳しくはセクション 12 をご覧ください。

表 6-2
レンジコマンド

コマンド[1]	説明
測定レンジコマンド：[2]	
smuX.measure.autorangei = smuX.AUTORANGE_ON	電流測定の自動レンジ切換えをオン
smuX.measure.autorangei = smuX.AUTORANGE_OFF	電流測定の自動レンジ切換え解除
smuX.measure.autorangev = smuX.AUTORANGE_ON	電圧測定の自動レンジ切換えをオン
smuX.measure.autorangev = smuX.AUTORANGE_OFF	電圧測定の自動レンジ切換え解除
smuX.measure.lowrangei = lowrange	自動切換え可能な最低電流測定レンジを設定
smuX.measure.lowrangev = lowrange	自動切換え可能な最低電圧測定レンジを設定
smuX.measure.rangei = rangeval	手動電流測定レンジを選択
smuX.measure.rangev = rangeval	手動電圧測定レンジを選択
ソースレンジコマンド：[2]	
smuX.source.autorangei = smuX.AUTORANGE_ON	電流ソースレンジの自動レンジ切換えをオン
smuX.source.autorangei = smuX.AUTORANGE_OFF	電流ソースの自動レンジ切換え解除
smuX.source.autorangev = smuX.AUTORANGE_ON	電圧ソースの自動レンジ切換えをオン
smuX.source.autorangev = smuX.AUTORANGE_OFF	電圧ソースの自動レンジ切換え解除
smuX.source.limiti = level	電圧ソースの電流リミットを設定
smuX.source.limitv = level	電流ソースの電圧リミットを設定
smuX.source.lowrangei = lowrange	自動切換え可能な最低電流ソースレンジを設定
smuX.source.lowrangev = lowrange	自動切換え可能な最低電圧ソースレンジを設定
smuX.source.rangei = rangeval	手動電流ソースレンジを選択
smuX.source.rangev = rangeval	手動電圧ソースレンジを選択

1. 2601 型：smuX = smua、2602 型：smuX = smua (Channel A) または smub (Channel B)
2. 使用可能なレンジについては表 6-1（p.6-2）をご覧ください。

レンジ設定プログラミングの例

次に示すのはソースと測定レンジ両方をコントロールするプログラミング例です。このプログラムにより、ソースメータは次のように設定されます。

- 電圧ソースレンジ：10V
- 電流測定レンジ：自動
- 測定の低域レンジ =10 μ A
- 電圧ソースの電流リミット：10mA

```
smua.reset()                                --260x 型のデフォルト設定呼び戻し
smua.source.rangev = 10                     --電圧ソースレンジを 10V に設定
smua.measure.autorangei =                   --電流測定の自動レンジ機能をオン
smua.AUTORANGE_ON
smua.measure.lowrangei = 1e-5               --最低レンジを 10μA に設定
smua.source.limiti = 1e-2                   --リミットレベルを 10mA に設定
```

桁数

測定値の表示分解能は DIGITS の設定により変化します。この設定はグローバルであり、ここで指定した桁数設定はすべての測定機能の表示分解能に共通に適用されます。

DIGITS で指定する桁数はリモート読み取りの形式には影響を与えません。また、表示桁数は測定の確度や速度にも影響を与えません。確度や速度は SPEED の設定によってコントロールされます（「速度」（p.6-8）参照）。

表示分解能の設定

表示分解能を設定するには、希望の桁数が表示されるようになるまで DIGITS キーを何回か押してください。表示分解能は 4.5、5.5、6.5 桁の順に巡回的に変化します。

> *注　2602 型をデュアルチャンネル表示モードで使用する場合の最大表示分解能は 4.5 桁です。2602 型をシングルチャンネル表示モードで使用する場合、DIGITS キーによる設定は非表示チャンネルに影響を与えません（非表示チャンネルの変更も推奨するメッセージが表示されます）。*

表示桁数のリモートプログラミング

桁数コマンド

表示桁数をコントロールするコマンドをまとめて表 6-3 に示します。更に詳しくはセクション 12 をご覧ください。

表 6-3
桁数コマンド

コマンド[1]	説明
display.smuX.digits = display.DIGITS_4_5 display.smuX.digits = display.DIGITS_5_5 display.smuX.digits = display.DIGITS_6_5	表示を 4.5 桁に設定 表示を 5.5 桁に設定 表示を 6.5 桁に設定

1. 2601 型：smuX = smua、2602 型：smuX = smua (Channel A) または smub (Channel B)

桁数プログラミングの例

```
smua.reset()                                --260x 型のデフォルト設定呼び戻し
display.smua.digits = display.DIGITS_5_5 --5.5 桁表示を選択
```

速度

A/D 変換器の積分時間（測定アパーチャとも呼びます、すなわち入力信号を測定する時間）を設定するには SPEED キーを使用します。積分時間はデータの有効桁数や読み取り時のノイズ、装置の最終的な読み取り速度に影響を与えます。積分時間は電源周波数から導かれるパラメータ NPLC(Number of Power Line Cycles) を用いて指定します、すなわち、60Hz ならば 1 PLC は 16.67msec（1/60）となり、50Hz ならば 1 PLC は 20msec（1/50）です。

一般的には最も短い積分時間（FAST ; 0.01 PLC）を使用すれば読み取り時間が短くなりますが、同時に読み取りノイズの増大と有効桁数の減少というデメリットが伴います。逆に積分時間を最長（25 PLC）にすると最良のコモンモードおよびノーマルモードノイズ除去率が得られますが、読み取り速度は最も遅くなります。その中間の設定は速度とノイズの兼ね合いを考えて選択します。装置に電源を投入したときのデフォルト速度設定としては NORMAL（1 PLC）が選択されます。

速度設定

速度の設定には以下の選択項目を含む SPEED 設定メニューを使用します。

SPEED 設定メニュー

SPEED キーを押してメニューを表示させます（CONFIG メニューから導くことも可能）。

- **FAST** — 速度を 0.01 PLC、表示分解能を 4-1/2 桁に設定します。
- **MED** — 速度を 0.10 PLC、表示分解能を 5-1/2 桁に設定します。
- **NORMAL** — 速度を 1.00 PLC、表示分解能を 6-1/2 桁に設定します。
- **HI ACCURACY** — 速度を 10.00 PLC、表示分解能を 6-1/2 桁に設定します。
- **OTHER** — 速度を 0.001 ～ 25 PLC の任意の値に設定したいときに使用します。このオプションを使用して速度を設定しても表示分解能設定は変化しません。

注　SPEED 設定はすべての測定機能に影響を与えます。速度設定後に表示分解能を変更したい場合は DIGITS キーを使用して変更することができます。2602 型をシングルチャンネル表示モードで使用しているときに SPEED キーで非表示チャンネルを設定変更しようとすると、チャンネルを切換えてから速度設定するように促すメッセージが表示されます。

速度設定のリモートプログラミング

速度コマンド

速度コントロールに使用するコマンドをまとめて表 6-4 に示します。更に詳しくはセクション 12 をご覧ください。

表 6-4
速度コマンド

コマンド[1]	説明
smuX.measure.nplc = nplc	速度を設定（nplc = 0.001 ～ 25）[2]

1. 2601 型：smuX = smua、2602 型：smuX = smua (Channel A) または smub (Channel B)
2. 速度は装置全体に共通ですから、すべての測定機能に影響を与えます。

速度設定プログラミングの例

速度を設定するには NPLC コマンドを使用します。例えば、速度を 10 PLC に設定したい場合は次のパラメータを送ります。

```
smua.reset()                    --260x 型のデフォルト設定呼び戻し
smua.measure.nplc = 10          --NPLC を 10 に設定
```

リラティブ

オフセットをゼロにする、あるいは現在および将来の読み取り値からベースライン読み取り値を引き去るために、相対測定機能（リラティブ）を使用します。REL をオンにすると、それ以後は実際の入力値から次式に従って リラティブ 値を引き去った値が表示されます。

表示される指示値 ＝ 実際の入力 - リラティブ 値

ある測定機能について決定した リラティブ 値はすべてのレンジについて共通です。例えば、6V レンジで使用中に リラティブ 値が 5V にセットされたとすれば、1V レンジや 100mV レンジに切換えた後も リラティブ 値は 5V です。

あるレンジに収まらないような リラティブ 値がセットされているとしても、そのレンジのオーバーフロー条件が変化することはなく、またそのレンジの許容入力範囲が広がるということもありません。6V レンジを例にとれば、リラティブ値の如何に関わらず入力が 6.06V を超えるとオーバーフローが発生します。

> ***注*** *リラティブ 値が確定すると REL ランプが点灯します。リラティブ 値は測定機能を切換えることにより無効になります。*

前面パネルからの リラティブ 操作

リラティブ 値の有効 / 無効

REL キーを押して リラティブ 値を確定させることによりゼロオフセット値のゼロクリア、またはベースラインゼロを決定することができます。REL キーを押したときの読み取り値（これが リラティブ 値になります）は自分自身からも引き算されますから、指示値はゼロになります。もう一度 REL キーを押すと リラティブ 機能が解除されます。

リラティブ 値を定義する

前面パネルから以下の操作を行うことで、現在選択されている測定機能に一意の リラティブ 値を確定させることができます。

1. まず CONGIF、続いて REL を押します。
2. いずれかの測定機能（CURRENT、VOLTAGE、OHMS、WATTS）を選択し、ENTER または回転ノブを押します。
3. 現在の リラティブ 値が表示されます。
4. 希望する リラティブ 値をセットします。
5. 希望する リラティブ 値がディスプレイに表示された状態で ENTER または回転ノブを押し、さらに EXIT を押してメニューから脱出します。

リラティブ のリモートプログラミング

リラティブ コマンド

リラティブ コマンドを表 6-5 にまとめて示します。

表 6-5
リラティブ コマンド

コマンド[1]	説明
リラティブ 値の設定	
smuX.measure.rel.leveli = relval	電流 リラティブ 値を設定
smuX.measure.rel.levelp = relval	パワー リラティブ 値を設定
smuX.measure.rel.levelr = relval	抵抗 リラティブ 値を設定
smuX.measure.rel.levelv = relval	電圧 リラティブ 値を設定
リラティブ の有効 / 無効切換え	
smuX.measure.rel.enablei = smuX.REL_OFF	電流 リラティブ 値を無効化
smuX.measure.rel.enablep = smuX.REL_OFF	パワー リラティブ 値を無効化
smuX.measure.rel.enabler = smuX.REL_OFF	抵抗 リラティブ 値を無効化
smuX.measure.rel.enablev = smuX.REL_OFF	電圧 リラティブ 値を無効化
smuX.measure.rel.enablei = smuX.REL_ON	電流 リラティブ 値を有効化
smuX.measure.rel.enablep = smuX.REL_ON	パワー リラティブ 値を有効化
smuX.measure.rel.enabler = smuX.REL_ON	抵抗 リラティブ 値を有効化
smuX.measure.rel.enablev = smuX.REL_ON	電圧 リラティブ 値を有効化

1. 2601 型：smuX = smua、2602 型：smuX = smua (Channel A) または smub (Channel B)

リラティブ のリモートプログラミング例

```
smua.reset()                          --260x 型のデフォルト設定呼び戻し
smua.measure.rel.leveli = 0.1         --電流 リラティブ 値を 100mA に設定
smua.measure.rel.enablei =            --電流 リラティブ 値を有効化
smua.REL_ON
```

フィルタ

フィルタ応答をうまく設定することによりノイズの多い測定を安定化することができます。ソースメータは読み取り値に演算を施すデジタルフィルタを使用しており、表示 / 保存 / 伝送される指示値は複数回（1 ～ 100）の読み取り値に平均化処理を施した値です。

フィルタタイプ

平均フィルタのタイプには「反復」と「移動」の 2 つのタイプがあります（図 6-1）。反復フィルタ（電源投入時のデフォルト設定）はスタックに読み取り値を積み上げてゆき（スタックの深さ＝ filter count）、それらの値の平均値を指示値とします。その後スタックを一度空にしてから次回の処理を開始します。

移動平均フィルタは先入れ先出し（“first-in, first-out”）スタックを使用します。スタック（深さ＝ filter count）がいっぱいになると読み取り値を平均化して 1 つの指示値を計算します。それ以後は新しい読み取り値が得られる度にスタックにそれを追加すると同時に一番古い読み取り値を捨て、スタックの平均値を再計算してそれを指示値とします。

フィルタを最初にオンにしたときのスタックは空になっています。したがって、一度スタックが満杯になるまではフィルタ処理値が得られないことになります。実際には最初の読み取り値を空のスタックに書き込むときに、同じ値をスタックの他の場所にもコピーしてスタックを満たすという操作を行います。このため、最初のフィルタ出力は最初の読み取り値と一致します。この方法によって最初から通常の移動平均計算が実行されることになりますが、スタックが実際の読み取り値で完全に満たされるまでは真の平均値は得られないことに注意が必要です。例えば、図 6-1A に示す例ではスタックを新しい読み取り値で満たすには 10 回の読み取り動作が必要になります。最初の 9 回のフィルタ処理値はコピーされた値を含む平均値です。

応答時間についての注意事項

フィルタの平均化モードとカウント数は全体としての読み取り速度に影響を与えます。反復フィルタは平均値の計算ごとにスタックをクリアしてその度に読み取り値を蓄積しなければなりませんから、移動平均フィルタよりもはるかに低速です。また、平均化する読み取り値の数も速度に影響を与え、その数が増えるごとに指示値の変化速度が低下します。

前面パネルからのフィルタコントロール

フィルタの設定

フィルタのタイプとカウント数を指定するにはフィルタ設定メニューを使用します。こうして指定されたフィルタの諸設定はすべての測定機能で共通に使用されます。

フィルタ設定メニュー

まず CONFIG を押し、続いて FILTER を押してフィルタ設定メニューを表示させます。

- AVERAGE-TYPE: このメニュー項目でフィルタのタイプ（MOVING / REPEAT）を選択します。
- AVERAGE-COUNT: このメニュー項目でフィルタのカウント数（読み取り値 1 ～ 100）を指定します。

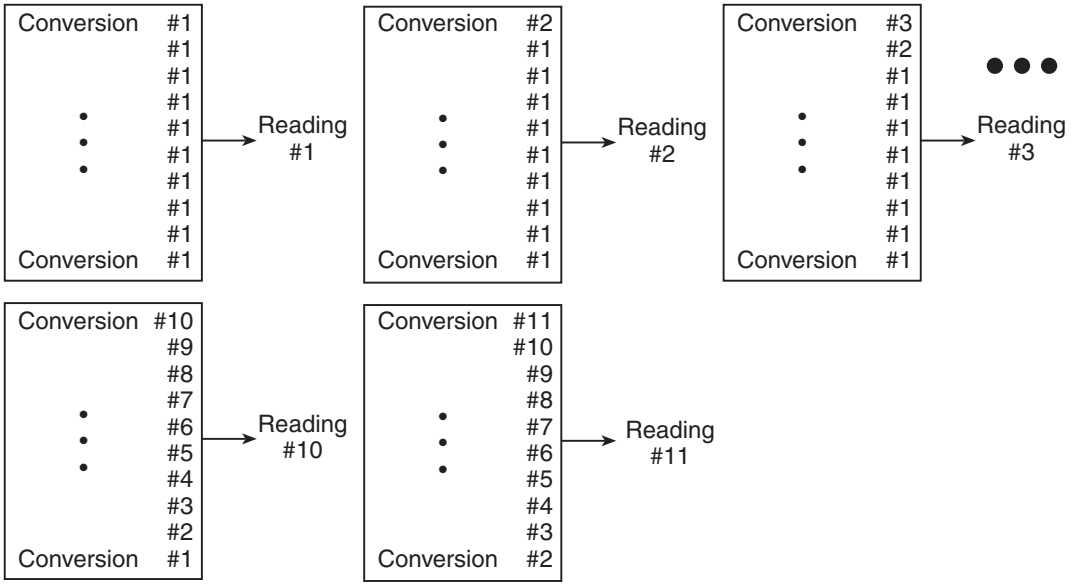
フィルタを使用可能にする

FILTER キーを押すことによりフィルタの機能がアクティブになり、フィルタが機能している間は FLT ランプが点灯しています。もう一度 FILTER を押すとフィルタの機能がオフになります。

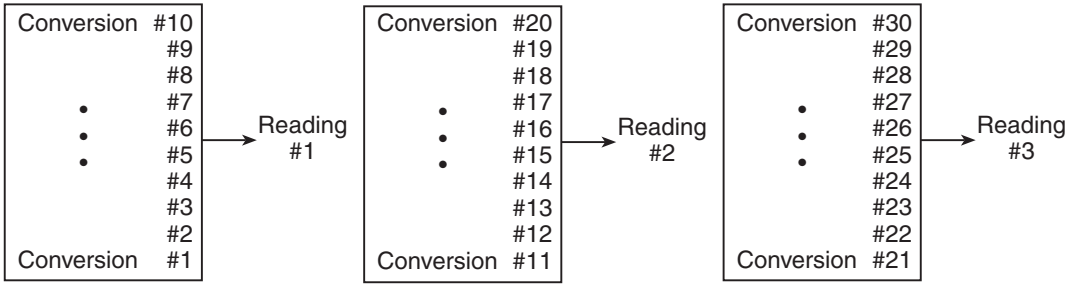
応答時間

フィルタのパラメータは、表示までに要する時間やデータの保存、フィルタ処理後の指示値出力等で必要となる速度と確度のトレードオフを考えて決める必要があります。速度と確度の兼ね合い、および入力信号変化に対する応答を考えることで読み取り値の変換回数が変わってきます。

図 6-1
移動平均フィルタと反復フィルタ



A. Type - Moving Average, Readings = 10



B. Type - Repeating, Readings = 10

フィルタのリモートプログラミング

フィルタコマンド

フィルタをコントロールするコマンドをまとめて表 6-6 に示します。更に詳しくはセクション 12 をご覧ください。

表 6-6
フィルタコマンド

コマンド	説明
smuX.measure.filter.count = count	フィルタカウントを設定（1 ～ 100）
smuX.measure.filter.enable = smuX.FILTER_ON	フィルタを動作状態にする
smuX.measure.filter.enable = smuX.FILTER_OFF	フィルタを停止
smuX.measure.filter.type = smuX.FILTER_MOVING_AVG	移動平均フィルタを選択
smuX.measure.filter.type = smuX.FILTER_REPEAT_AVG	反復フィルタを選択

1. 2601 型：smuX = smua、2602 型：smuX = smua (Channel A) または smub (Channel B)

フィルタのプログラミング例

次に示す例ではフィルタ特性を次のように設定します。

- フィルタタイプ：移動平均
- フィルタカウント：10
- フィルタの状態：動作中

```
smua.reset()                                        --260x 型のデフォルト設定呼び戻し
smua.measure.filter.count = 10                      --フィルタカウントを 10 に設定
smua.measure.filter.type = smua.FILTER_MOVING_AVG   --移動平均フィルタ選択
smua.measure.filter.enable = smua.FILTER_ON         --フィルタを動作状態にする
```

7 バッファ（データ保存）

セクション 7 のトピックス

概要 , 7-2

データ保存のまとめ , 7-2

前面パネル操作によるデータ保存 , 7-2
- バッファの設定 , 7-2
- バッファ設定メニュー , 7-3
- 読み取り値の保存 , 7-3
- 読み取り値の呼び出し , 7-4

リモート操作によるデータ保存 , 7-5
- データ保存コマンド , 7-5
- バッファの読み出し , 7-6
- 時間と日付の値 , 7-10
- バッファステータス , 7-10
- バッファの動的割り付け , 7-11
- バッファのプログラミング例 , 7-12

概要

このセクションではデータ保存に使用するバッファ領域の使用法を次の項目に分けて詳しく説明します。

- 「データ保存のまとめ」、p.7-2
- 「前面パネル操作によるデータ保存」、p.7-2
- 「リモート操作によるデータ保存」p.7-5

データ保存のまとめ

ソースメータはチャンネル毎に 2 つのバッファを持ち、それぞれに 1 から始まって 100,000 個以上までの読み取り値を保存することができます。データ保存動作中の装置はディスプレイに表示された値をバッファに保存します。バッファに書き込まれる読み取り値には番号が付けられるとともに、対応するソース値とタイムスタンプも同時に保存されます。

前面パネル操作によるデータ保存

バッファの設定

バッファの設定にはバッファ設定メニューを使用します。このメニューへのアクセス方法は次のとおりです。

1. CONFIG キーを押し、続いて STORE キーを押します。
2. 以下のメニュー項目を必要に応じて選択をしバッファを設定します。

注	*データ要素（ソース値またはタイムスタンプ）の保存を開始または停止するときは、その前にバッファをクリアしなければなりません。*

バッファ設定メニュー

バッファ設定メニューには次のようなメニュー項目が含まれます。

- **COUNT:** 保存する読み取り値の数を設定します（1 ～ 110,000）。
- **CHANA_BUFF:** チャンネル A のバッファを設定します。
  - **DEST:** データの保存先を指定します（Buffer 1、Buffer 2、または NONE）。
  - **BUFFER1:** バッファ 1 を設定します。
    - **CLEAR:** バッファをクリアします（YES または NO）。
    - **ELEMENTS:** データ保存の対象となる以下の要素の保存（ON）/非保存（OFF）を指定します。SRC-VAL（ソース値）または TSTAMP（タイムスタンプ）。
  - **BUFFER2:** バッファ 2 を設定します。
    - **CLEAR:** バッファをクリアします（YES または NO）。
    - **ELEMENTS:** データ保存の対象となる以下の要素の保存（ON）/非保存（OFF）を指定します。SRC-VAL（ソース値）または TSTAMP（タイムスタンプ）。
- **CHANB_BUFF:** チャンネル B のバッファを設定します（2602 型のみ）。
  - **DEST:** データの保存先を指定します（Buffer 1、Buffer 2、または NONE）。
  - **BUFFER1:** バッファ 1 を設定します。
    - **CLEAR:** バッファをクリアします（YES または NO）。
    - **ELEMENTS:** データ保存の対象となる以下の要素の保存（ON）/非保存（OFF）を指定します。SRC-VAL（ソース値）または TSTAMP（タイムスタンプ）。
  - **BUFFER2:** バッファ 2 を設定します。
    - **CLEAR:** バッファをクリアします（YES または NO）。
    - **ELEMENTS:** データ保存の対象となる以下の要素の保存（ON）/非保存（OFF）を指定します。SRC-VAL（ソース値）または TSTAMP（タイムスタンプ）。

注　2601 型のバッファ設定メニューの項目も、チャンネル選択機能が無いことを除いて上記と同様です。

読み取り値の保存

注　2602 型の STORE キーを押すと、NONE（上記参照）が選択されていない限りは両方のチャンネルの保存先バッファが上書きされます。

読み取り値を保存するには以下のステップを実行してください。

1. 上に説明した設定メニューを使用して、ソースメータを希望する設定内容とバッファ構成に合わせてセットアップします。
2. 出力を ON にします。
3. STORE キーを押します。データ保存実行中であることを示すアスタリスク（*）ランプが点灯します。データ保存完了とともにランプが消灯します。
4. 完了前にデータ保存を中断したい場合は EXIT を押してください。

読み取り値の呼び出し

バッファに保存された読み取り値をディスプレイに表示させるには RECALL キーを押してください。RECALL を繰り返して押すことにより、まずチャンネル A の Buffer 1、続いて Buffer 2 の内容がスキャンされ、さらに続いてチャンネル B（2602 型のみ）のバッファがスキャンされます。バッファが空であった場合はその旨を知らせるメッセージが表示されます。

ディスプレイの上の行の左寄りに読み取り値、右寄りにバッファの位置番号が表示されます。ディスプレイの下の行の左寄りにソース値（イネーブルされている場合）が表示され、タイムスタンプ（使用されている場合）は右寄りに表示されます。RECALL キーを押してバッファを切換えると、ソース表示フィールドにバッファの識別マーカが次の順にしたがって表示されます。Src1A（Buffer 1、チャンネル A）→ Src2A（Buffer 2、チャンネル A）、2602 型の場合は更にこれに続いて Src1B（Buffer 1、チャンネル B）→ Src2B（Buffer 2、チャンネル B）。

バッファの位置番号

バッファの位置番号はソースメジャー読み取り値のバッファメモリ内での位置を示します。たとえば位置番号が #000001 であれば、そのソースメジャー読み取り値がメモリの先頭位置に保存されていることを表しています。

タイムスタンプ

タイムスタンプを記録する設定になっている場合は、バッファ内の最初のソースメジャー読み取り値（#0000001）に 0000000.001 秒というタイムスタンプが付きます。それ以後の読み取り値にはデータ保存開始時を起点とする経過時間がタイムスタンプとして記録されます。ある読み取り値から次の読み取り値までの時間は読み取り速度の設定に応じて変わります。

バッファ内の位置を指定して読み取り値を表示

バッファ内の別なソースメジャー読み取り値を表示させたい場合は、そのデータに対応するメモリ位置番号をディスプレイに表示させなければなりません。そのためには回転ノブまたは CURSOR キーを使用してカーソルを位置番号の希望する桁へ移動し、続いて回転ノブを使用してその桁の数字を増加 / 減少させることによって希望の位置番号を選択します。

保存データの呼び出しモードを終了するには EXIT キーを押してください。

リモート操作によるデータ保存

データ保存コマンド

データの保存操作に関連するコマンドをまとめて表 7-1 に示します。これらのコマンドの更に詳しい説明についてはセクション 12 をご覧ください。

表 7-1
データ保存コマンド

コマンド[1]	説明
smuX.nvbuffer1.clear()	Buffer 1 クリア
smuX.nvbuffer2.clear()	Buffer 2 クリア
mybuffer = smuX.makebuffer(n)	バッファ領域を動的に割り付け（読み取り値n個）
mybuffer = nil	動的に割り付けられたバッファ領域を削除
読み取り値保存コマンド:[2]	
smuX.measure.count = count	count で指定した数のバッファ読み取り値を保存
smuX.measure.overlappedi(rbuffer)	電流読み取り値をバッファに保存
smuX.measure.overlappediv(ibuffer, vbuffer)	電流と電圧読み取り値をそれぞれのバッファに保存（電流値、続いて電圧値が別々なバッファに保存されます）。
smuX.measure.overlappedp(rbuffer)	パワー読み取り値をバッファに保存
smuX.measure.overlappedr(rbuffer)	抵抗読み取り値をバッファに保存
smuX.measure.overlappedv(rbuffer)	電圧読み取り値をバッファに保存
読み取り値へのアクセスコマンド:[2]	
print(smuX.measure.i(rbuffer))	バッファに保存されている最初の電流読み取り値を返す
print(smuX.measure.iv(ibuffer, vbuffer))	2 つの別々なバッファから最初の電流読み取り値と電圧読み取り値を返す
print(smuX.measure.p(rbuffer))	バッファに保存されている最初のパワー読み取り値を返す
print(smuX.measure.r(fbuffer))	バッファに保存されている最初の抵抗読み取り値を返す
print(smuX.measure.v(rbuffer))	バッファに保存されている最初の電圧読み取り値を返す
printbuffer(start_index, end_index, st_1 [, st_n])	バッファの部分テーブルを指定してデータを出力する start_index（出力する値の開始インデックス） end_index（出力する値の終了インデックス） st_1 … st_n（出力元となる部分テーブル）[3]
printnumber(v1[, vn])	指定したバッファ形式で数値を出力 v1 … vn（出力する数字）

1. 2601 型：smuX = smua、2602 型：smuX = smua (Channel A) または smub (Channel B)
2. rbuffer、ibuffer、vbuffer = smuX.nvbuffer1 または smuX.nvbuffer2
3. さらに詳しくは「バッファの読み出し」（p.7-6）をご覧ください。

バッファの読み出し

読み取り値は同期式やオーバーラップ式を含む何通りもの方法で取得が可能です。さらに、1 点測定用ルーチンの設定を変えて多点測定用とし、通常は 1 点測定のみを行うという方法も可能です。読み取りで得られるのは測定値ばかりではなく、個々の読み込みに対応した測定ステータス（“In Compliance”または“Overranged”）もデータ要素として取り込まれます。

測定値を返すすべてのルーチンはこれらの値を読み込みバッファとして返します。オーバーラップ測定の場合は常に測定値が読み込みバッファに書き込まれて返されます。同期測定の場合は 1 点の測定値を返すか、または読み込みバッファとして返します。読み込みバッファには追加情報も書き込まれていますから、使用法に馴れたユーザならば追加情報にアクセスしてデータを活用することができます。

読み込みバッファは LUA テーブルに基いて構成されており、測定値自体は通常の配列と同じ方法でアクセスすることができます。読み込みバッファが rb であったとすれば、最初の測定値は rb[1]、9 番目の測定値は rb[9] としてアクセスします。テーブル内の追加情報はそのテーブルの追加メンバーとしてアクセスします。

読み込みバッファの名称

2 つの領域が不揮発性読み込みバッファとして用意されており、smuX.nvbuffer1（Buffer 1）および smuX.nvbuffer2（Buffer 2）という名前で区別されます。特定のバッファへアクセスするには使用するコマンドにそのバッファの属性値を与えます。たとえば、次のコマンドはチャンネル A から取得した電流読み取り値を Buffer 1 へ保存します。

```
smua.measure.overlappedi(smua.nvbuffer1)
```

バッファへの保存をコントロールする属性

バッファの保存に関する属性をまとめて表 7-2 に示します。バッファのパラメータへアクセスする際に必要となる読み出し専用属性をまとめて表 7-3 に示します。バッファにどの要素を保存するかをコントロールするには、対象となるバッファに必要な属性を割り付ける必要があります。その例としてチャンネル A の Buffer 1 をコントロールする例を表 7-4（p.7-8）に、読み出し専用属性のプログラム例を表 7-5（p.7-8）に示します。

> ***注　　バッファのコントロール属性を変更しようとするときは、必ずその前に smuX.nvbufferX.clear() コマンドを使用してバッファをクリアしてください。***

表 7-2
バッファへの保存をコントロールする属性

保存属性	説明
appendmode	アペンドモードは ON または OFF（デフォルト）のいずれかの状態をとります。このモードを OFF にしておくと、新しい測定値が今までの内容を上書きしながらバッファに保存されます。このモードを ON にしておくと、それまで rb[n+1] であった位置に新しい測定の最初の値が書き込まれます。
collectsourcevalues	このモードを ON にすると、ソース値が読み取り値といっしょにバッファに保存されます。このため、1 個の読み取り値あたり 4 バイトが余分に必要となります。この値は ON または OFF（デフォルト）のいずれかをとり、変更できるのはバッファが空のときだけです。
collecttimestamps	この属性を ON に設定すると、タイムスタンプが読み取り値といっしょにバッファに保存されます。このため、1 個の読み取り値あたり 4 バイトが余分に必要となります。この値は ON または OFF（デフォルト）のいずれかをとり、変更できるのはバッファが空のときだけです。
timestampresolution	タイムスタンプの時間分解能（秒）を示します。デフォルト分解能は 0.000001 秒に設定されています。この分解能を指定すると、バッファに最高 71 分までの相互に重複しないタイムスタンプを保存することができます。長時間を要する試験の場合はこの値を大きくしてください。注：分解能の最小設定は 1 μs（0.000001 秒）です。

表 7-3
バッファの読み出し専用属性

保存属性	説明
basetimestamp	rb[1] に読み取り値を書き込んだ時刻の、装置への電源投入を起点としたタイムスタンプ。
capacity	読み取りバッファに保存可能な読み取り値の総数。
n	読み取りバッファに書き込まれている読み取り値の数。

表 7-4
バッファをコントロールするプログラミング例

コマンド	説明
`smua.nvbuffer1.collectsourcevalues = 1`	ソース値の保存をオンにします。
`smua.nvbuffer1.appendmode = 1`	バッファをアペンドモードに設定します。
`smua.nvbuffer1.collecttimestamps = 0`	タイムスタンプの保存をオフにします。
`smua.nvbuffer1.timestampresolution = 0.001`	タイムスタンプの時間分解能を 0.001 秒に設定します。

表 7-5
バッファの読み取り専用属性をコントロールするプログラミング例

コマンド	説明
`number = smua.nvbuffer1.n`	バッファ内の読み取り値の数を要求します。
`buffer_size = smua.nvbuffer1.capacity`	バッファサイズを要求します。

バッファの読み取り属性

バッファから呼び出す要素をコントロールするための属性をまとめて表 7-6 に示します。特定の要素にアクセスするには、バッファを指定するときに希望する属性値を付け加えます。

たとえば、次のコマンドは Buffer 1 から 100 個のチャンネル A 読み取り値を返します。

```
printbuffer(1, 100, smua.nvbuffer1.readings)
```

同じように、次の例では Buffer 1 から 100 個のチャンネル A ソース値を返します。

```
printbuffer(1, 100, smua.nvbuffer1.sourcevalues)
```

readings は読み取り値を取得するためのデフォルト属性値ですが、これは省力可能です。たとえば、次のコマンドでは readings が省略されていますが、同じように Buffer 1 から 100 個のチャンネル A 読み取り値を返します。

```
printbuffer(1, 100, smua.nvbuffer1)
```

表 7-6
呼び出し属性

呼び出し属性[1]	説明
measurefunctions	測定値読み取りの対象となった機能（電流 / 電圧 / 抵抗 / ワット）を示す文字列配列（LUA テーブル）。
measureranges	測定実施時に使用した測定レンジのフルスケール値を保存した配列（LUA テーブル）。
readings	読み取りバッファに保存されている読み取り値を保存した配列（LUA テーブル）。この配列は読み取りバッファを直接アクセスしたのと同じデータを保持します。たとえば、rb[2] と rb.readings[2] は同じ値です。
sourcefunctions	測定実行時のソース機能を示す文字列（Current または Voltage）を保存した配列（LUA テーブル）。
sourceoutputstates	ソースの状態を示す文字列（Off または On）を保存した配列（LUA テーブル）。
sourceranges	測定実施時に使用したソースレンジのフルスケール値を保存した配列（LUA テーブル）。
sourcevalues	ソース機能をオンにした場合、読み取り時に機能していたソース値がこの配列（LUA テーブル）に保存されます。
statuses	バッファ内のすべての読み取り値のステータス値を保存した配列（LUA テーブル）。ステータス値を浮動少数点にエンコードした値が書き込まれます（表 7-7 参照）。
timestamps	読み取り実行時のタイムスタンプ（秒単位）を保存した配列（LUA テーブル）。これらの値は、そのバッファの basetimestamp を起点とする相対値です（表 7-3）。

1. デフォルト属性値 readings は省力可能です。たとえば、smua.nvbuffer1 と smua.nvbuffer1.readings は両方共 Buffer 1 のチャンネル A 読み取り値を返します。

時間と日付の値

時間と日付に関する値はすべて装置に電源を入れた時刻を起点とする秒単位の値で表現され、os.clock() 関数を使用すればこの形式で読み出すことができます。

時間を秒数で表現する形式を「標準時間形式」と呼びます。TSL は浮動小数点数であるため、電源投入時を起点とする時間および日付の値の精度は、一般に通電時間が長くなるほど低下します。精度の低下は全経過時間に対して 0.06ppm 程度の値になります。この精度は一般に装置の時間ベースよりもはるかに高く、実際上の問題を引き起こすものではありませんが、ユーザが殆ど気付かないタイムベースのドリフトと比較して明瞭に現われる要素であるため注意が必要です。

バッファステータス

バッファの読み取りステータスには表 7-7 に示すステータス情報が数値として組み込まれています。ステータス情報を読み取るには次のコマンドを送ります。

```
stat_info = smua.nvbuffer1.statuses[2]
```

表 7-7
バッファのステータスビット

ビット	名称	16 進表記	説明
B0	TBD	0x01	未使用
B1	Overtemp	0x02	過温度条件
B2	AutoRangeMeas	0x04	測定レンジに自動レンジ機能適用
B3	AutoRangeSrc	0x08	ソースレンジに自動レンジ機能適用
B4	4Wire	0x10	4 線式（リモート）センシングモード ON
B5	Rel	0x20	読み取り値に リラティブ を適用
B6	Compliance1	0x40	ソース機能がコンプライアンス条件を満たしている
B7	Filtered	0x80	読み取り値にフィルタ処理を実施

バッファの動的割り付け

読み込みバッファ（RAM）は動的割り付けが可能です。バッファの作成と割り付けには smuX.makebuffer(n) コマンドを使用します。ここに、n はバッファが保存可能な読み取り値の数を表します。たとえば、次のコマンドはチャンネル A 用に mybuffer という名前で 100 個の読み取り値を保存できるバッファ領域を確保します。

```
mybuffer = smua.makebuffer(100)
```

割り付け済みのバッファを削除するには次のコマンドを使用します。

```
mybuffer = nil
```

動的に割り付けられたバッファは smuX.nvbufferY バッファと区別せずに使用することができます。後者の説明については「読み込みバッファの名称」（p.7-6）をご覧ください。

バッファのプログラミング例

定義済みバッファの例

次に示すのはチャンネル A 用として定義済みの Buffer 1 を使用してデータを保存するプログラム例です。ソースメータは 0.01V から 1 V まで 0.01V ステップ で電圧を発生させるループ動作（実質的に階段スイープ）を行い、そのとき得られた 100 個の電流読み取り値とソース値を Buffer 1 に書き込みます。それに続いて 100 個全部の読み取り値とソース値を呼び出します。

```
smua.reset()                                    --260x 型のデフォルト設定呼び戻し
display.screen = 0                              --表示対象をチャンネル A に指定
display.smua.measure.func =                     --表示電流
display.MEASURE_DCAMPS
smua.measure.autorangei = smua.AUTORANGE_ON     --電流測定の自動レンジ機能選択
format.data = format.ASCII                      --ASCII データ形式を選択
smua.nvbuffer1.clear()                          --Buffer 1 クリア
smua.nvbuffer1.appendmode = 1                   --バッファをアペンドモードに設定
smua.nvbuffer1.collectsourcevalues = 1          --ソース値の保存をオン
smua.measure.count = 1                          --count 値を 1 にセット
smua.source.func = smua.OUTPUT_DCVOLTS          --電圧ソース機能を選択
smua.source.levelv = 0.0                        -- バイアス電圧を 0V に設定
smua.source.output =smua.OUTPUT_ON              --出力 ON
for v = 0.01, 1.0, 0.01 do                      --ループ動作で電圧を 0.01 から 1V まで発生

   smua.source.levelv = v                       --ソース電圧設定
   smua.measure.i(smua.nvbuffer1)               --電流を測定してバッファに保存
   waitcomplete()                               --読み込み完了待ち
end                                             --ループ終了
smua.source.output =smua.OUTPUT_OFF             --出力 OFF
printbuffer(1, 100, smua.nvbuffer1.readings)    --読み込み値 1 ～ 100 を返す
printbuffer(1, 100,                             --ソース値 1 ～ 100 を返す
smua.nvbuffer1.sourcevalues)
```

デュアルバッファの使用例

次に示すのは両方のバッファを使用して電流読み取り値を Buffer 1 に、電圧読み取り値を Buffer 2 へ保存するプログラム例です。ソースメータは 100 個の電流と電圧読み取り値を保存し、それに続いてこれら 100 セットの読み取り値を呼び出します。

```
smua.reset()                                          --260x 型のデフォルト設定呼び戻し
smua.measure.autorangei = smua.AUTORANGE_ON           --電流測定の自動レンジ機能選択
smua.measure.autorangev = smua.AUTORANGE_ON           --電圧測定の自動レンジ機能選択
format.data = format.ASCII                            --ASCII データ形式を選択
smua.nvbuffer1.clear()                                --Buffer 1 クリア
smua.nvbuffer2.clear()                                --Buffer 2 クリア
smua.measure.count = 100                              --バッファカウントを 100 に設定
smua.measure.interval = 0.1                           --測定間隔を 0.1 秒に設定
smua.source.func = smua.OUTPUT_DCVOLTS                --電圧ソース機能を選択
smua.source.levelv = 1                                --出力レベル 1V
smua.source.output =smua.OUTPUT_ON                    --出力 ON
smua.measure.overlappediv                             --電流読み取り値を Buffer 1、電圧読み
(smua.nvbuffer1,smua.nvbuffer2)                         取り値を Buffer 2 へ保存
waitcomplete()                                        --バッファがいっぱいになるのを待つ
smua.source.output =smua.OUTPUT_OFF                   --出力 OFF
printbuffer(1, 100, smua.nvbuffer1)                   --Buffer 1 の読み取り値 1 ～ 100 を返す
printbuffer(1, 100, smua.nvbuffer2)                   --Buffer 2 の読み取り値 1 ～ 100 を返す
```

動的に割り付けられたバッファを使用する例

次に示すのはチャンネル A 用として定義済みの mybuffer を使用してデータを保存するプログラム例です。ソースメータは 100 個の電流読み取り値を mybuffer に書き込み、それに続いて全部の読み取り値を呼び出します。

```
smua.reset()                                    --260x 型のデフォルト設定呼び戻し
smua.measure.autorangei = smua.AUTORANGE_ON     --測定に自動レンジ機能を使用
format.data = format.ASCII                      --ASCII データ形式を選択
mybuffer = smua.makebuffer(100)                 --100 個の読み取り値を保存する
                                                  mybuffer を割り付け
smua.measure.count = 100                        --バッファカウントを 100 に設定
smua.measure.interval = 0.1                     --測定間隔を 0.1 秒に設定
smua.source.func = smua.OUTPUT_DCVOLTS          --電圧ソース機能を選択
smua.source.levelv = 1                          --出力レベル 1V
smua.source.output =smua.OUTPUT_ON              --出力 ON
smua.measure.overlappedi(mybuffer)              --電流読み取り値を mybuffer に保存
waitcomplete()                                  --バッファがいっぱいになるのを待つ
smua.source.output =smua.OUTPUT_OFF             --出力 OFF
printbuffer(1, 100, mybuffer)                   --mybuffer から 100 個の読み取り値を
                                                  呼び出し
mybuffer = nil                                  --mybuffer 削除
```

8
ソースメジャーの概念

セクション 8 のトピックス

概要 , 8-2

コンプライアンス リミット , 8-2
- 最大コンプライアンス , 8-2
- コンプライアンスの原理 , 8-3

スイープ波形 , 8-4
- 階段スイープ , 8-4
- パルススイープ , 8-5

過熱保護 , 8-6
- 過熱予防に役立つパワー計算式 , 8-6
- パワー計算 , 8-6

動作の限界 , 8-7
- ソースかシンクか , 8-7
- 電流ソース（I -Source）の動作限界 , 8-8
- 電圧ソース（V -Source）の動作限界 , 8-12
- 電流ソース - 電流測定、電圧ソース - 電圧測定 , 8-16

基本回路構成 , 8-16
- 電流ソース , 8-16
- 電圧ソース , 8-17
- 測定のみ（V または I）, 8-18

ガード , 8-20
- ガードの概要 , 8-20
- ガードの接続 , 8-20

パルスの概念 , 8-23
- パルス周期 , 8-23
- パルスのライズ / フォールタイム , 8-23
- パルスのデューティサイクル , 8-24

概要

このセクションではソースメジャーの概念を以下の項目に分けて詳しく解説します。

- 「コンプライアンスリミット」、p.8-2
- 「スイープ波形」、p.8-4
- 「過熱保護」、p.8-6
- 「動作の限界」、p.8-7
- 「基本回路構成」、p.8-16
- 「ガード」、p.8-20
- 「パルスの概念」、p.8-23

コンプライアンス リミット

ソースメータを電圧ソースとして使用する場合は電流にリミット（100pA ～ 3A）を設定することができます。逆にソースメータを電流ソースとして使用する場合は電圧にリミット（100nV ～ 40V）を設定することができます。「コンプライアンスリミット」（p.4-3）で説明した条件が成立する場合を除けば、ソースメータ出力がコンプライアンスリミットを超えることはありません。

最大コンプライアンス

各ソースレンジに適用される最大コンプライアンス値をまとめて表 8-1 に示します。

表 8-1
最大コンプライアンス値

ソースレンジ	最大コンプライアンス値
100mV	3A
1V	3A
6V	3A
40V	1A
100nA	40V
1µA	40V
10µA	40V
100µA	40V
1mA	40V
10mA	40V
100mA	40V
1A	40V
3A	6V

コンプライアンスの原理

コンプライアンスはクランプ（制限回路）として機能します。すなわち、出力がコンプライアンス値に達すると、ソースメータは出力がその値を超えないように機能します。具体的にはソースが電圧ソースから電流ソースに（または電流ソースから電圧ソースに）切り換わってコンプライアンスを保とうとします。

その例として、次の条件を考えてみます。

ソースメータ：V_{SRC} = 10V; I_{CMPL} = 10mA

DUT 抵抗：10Ω

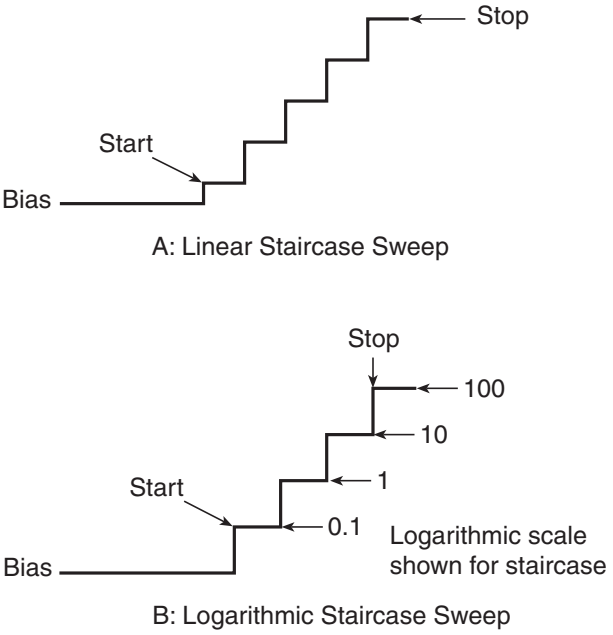
ソース電圧が 10V、DUT 抵抗が 10Ω ですから、DUT を流れる電流は 10V/10Ω = 1A ということになります。しかし、コンプライアンスが 10mA に設定されていることから電流はこの値を超えることができず、したがって抵抗両端の電圧は 100mV に制限されます。これは実効として、10V の電圧ソースが 100mV というコンプライアンス値を設定された 10mA 電流ソースに変換されたことになります。

スイープ波形

階段スイープ

階段スイープは線形階段スイープと対数階段スイープの基本的な 2 つのタイプに分かれます（図 8-1 参照）。線形階段スイープはスタートレベルからストップレベルまで全部等しいステップでリニアに変化します。対数階段スイープもこれによく似ていますが、1 ディケードあたり指定されたステップ数で対数スケールに従って変化する点が異なっています。スイープ操作の詳細についてはセクション 5 をご覧ください。

図 8-1
2 種類の基本スイープ波形



階段スイープを使用する代表的なアプリケーションとしては、たとえば 2 端子や 3 端子半導体デバイスの I-V 曲線や、電圧 - 漏洩電流特性、半導体の破壊特性測定などが挙げられます。

パルススイープ

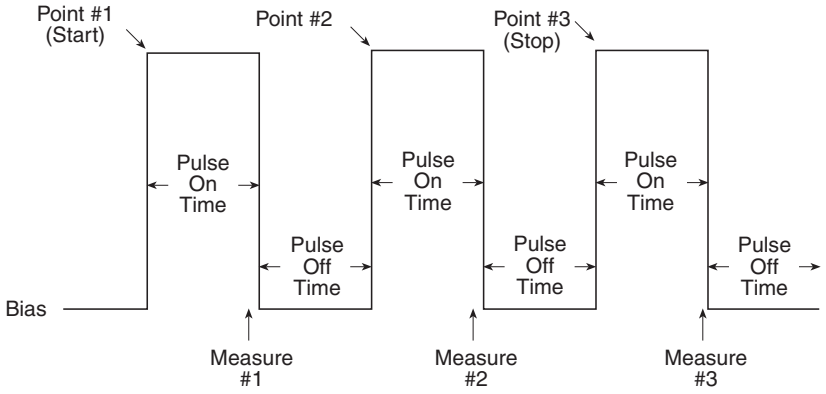
260x 型は次のようなパルススイープを実行する機能を備えています。

- 固定電圧パルス
- 固定電流パルス

パルススイープも前ページで説明した階段スイープに似ていますが、各スイープステップのソースレベルが一定レベルを持つのではなく、パルスである点が異なります。パルスモードスイープのそれぞれのタイプについて、パルス幅（オン時間）とパルスディレイ（オフ時間）をプログラムすることができます。3 個のパルスから構成したスイープ例を図 8-2 に示します。パルススイープについて更に詳しくは「パルススイープ」（p.5-6）および「パルスの概念」（p.8-23）をご覧ください。

パルススイープは熱応答の測定が必要となるアプリケーションや、外部の被測定物（DUT）への持続的なパワーの印加が破壊につながるような測定で使用されます。

図 8-2
パルススイープの例



過熱保護

ソースメータを過熱から防止するためには適切な換気が必要です。万一ソースメータが過熱状態になった場合は出力を遮断するための過熱保護回路を備えており、実際に出力が過熱によりトリップした場合は過熱条件を通知するメッセージが表示されます。一旦出力が遮断されると、装置温度が充分に下がるまで出力は復帰しません。

過熱予防に役立つパワー計算式

ソースメータが過熱する可能性の有無を判定するのにはパワー計算式が役立ちます。[1]
下式に現われる変数は次の意味を持っています。

I = 出力電流

V_{EXT} = 外部パワー発生源の電圧（第 2 および第 4 象限動作、図 8-3）

D_C[2] = デューティサイクル（0 ～ 1、たとえば 50% ならば 0.5）

T_{amb} = 周囲温度（0° ～ 50° C）

T_{der} = 30 ℃ を超えた差分温度（ソースメータ出力は 30 ℃ 以上 50 ℃ までの範囲で 1W/° C の割合でディレーティングされます）

パワー計算

40V と 6V レンジを使用するときは、次に示す計算式を用いてソースメータが過熱するか否かを判定することができます。

6V レンジ：

$$\left|(18 + V_{EXT})I\sqrt{D_C}\right| \le 55 - T_{der}$$

40V レンジ：

$$\left|(5.7 + V_{EXT})I\sqrt{D_C}\right| \le 55 - T_{der}$$

1. 計算式は両方のチャンネルの、パワーソースとシンクに同時に適用できます。
2. 1 未満のデューティサイクルを式に代入する場合は、最長オン時間を 10 秒未満としなければなりません。

動作の限界

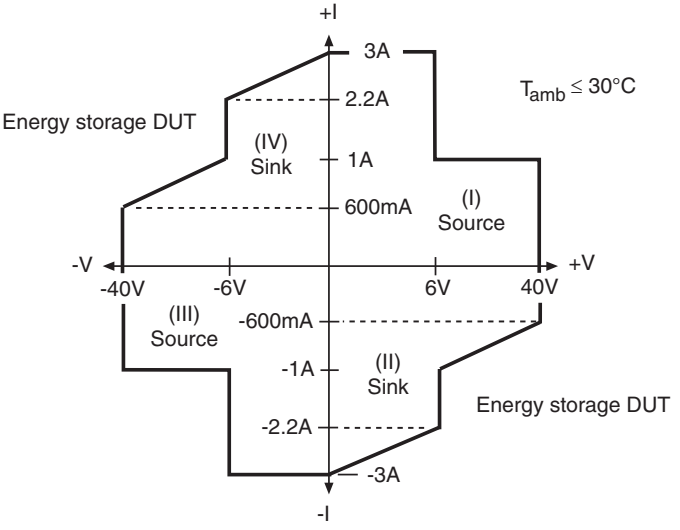
ソースかシンクか

どのようにプログラムされるか、および出力に何（負荷またはソース）を接続するかによって、ソースメータは図 8-3 に示す 4 象限のいずれでも動作が可能です。第 1（I）または第 3（III）象限で動作させると、ソースメータはソースとして機能（V と I が同一極性）し、ソースメータが負荷に対してパワーを供給します。

第 2（II）または第 4（IV）象限で動作させると、ソースメータはシンク（V と I が逆極性）として機能します。シンクとして動作する場合には外部にパワーを供給するのではなく内部でパワーを散逸しますから、外部のソースあるいはパワー貯蔵デバイス（キャパシタやバッテリ）がシンク領域での動作を駆動することになります。

図 8-3 に示すのは連続的にパワーを出力する場合の一般的な動作限界です（ディレーティングファクターについては「パワー計算」(p.8-6）の説明を参照)。この図においては、3A、6V および 1A、40V が通常のマグニチュードになります。この図ではスケールを調節して限界を描いてあることに注意してください。

図 8-3
パワーを連続的に変化させたときの動作限界



電流ソース（I -Source）の動作限界

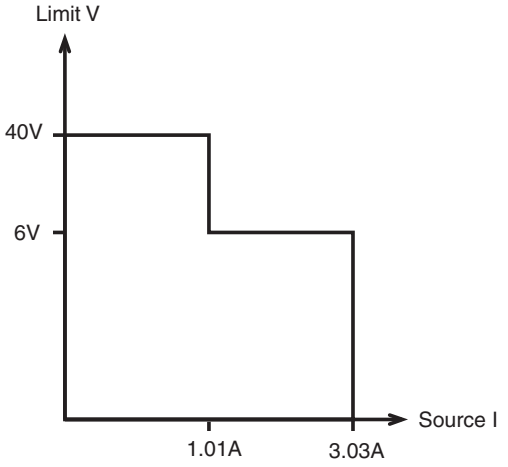
図 8-4 に示すのは I-Source の動作限界です。図に示されているのは第 1 象限の動作だけですが、他の 3 象限についてもほぼ同様に動作します。

図 8-4A に示すのは I-Source の出力特性です。図から分るとおり、ソースメータは 40V で 1.01A、6V で 3.03A を出力する能力を持っています。1.01A を超える電流を供給しようとすると電圧が 6V に制限されることに注意してください。

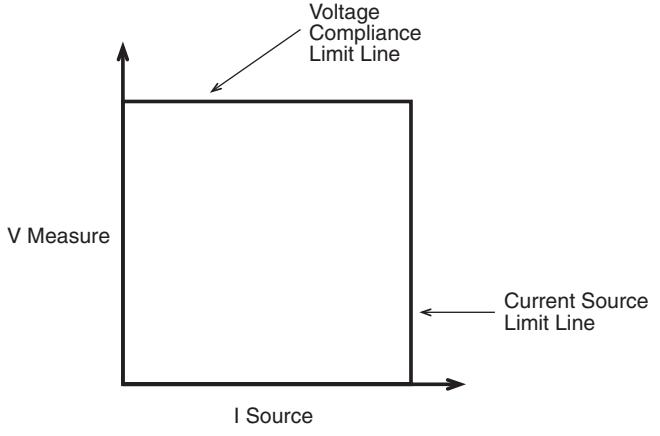
図 8-4B に示すのは I-Source の制限ラインです。電流ソースの制限ラインは現在選択されている電流ソースレンジで供給可能な最大ソース値を表しています。電圧コンプライアンス制限ラインは実効を持つ実際のコンプライアンスを表します。コンプライアンスはリアルとレンジの両方が可能であることに注意が必要です（「コンプライアンスリミット」、p.8-2、参照）。

これらの制限ラインはこの動作象限におけるソースメータの動作限界を表す限界です。これらの制限ラインの内部（またはライン上）であれば、どこにでも動作点を置くことができます。他の象限における限界もこれと良く似ています。

図 8-4
電流ソース（I -Source）の限界



A) 出力特性



B) 制限ライン

ソースメータが動作可能な限界は出力端子に接続されている負荷（DUT）に依存します。図 8-5 に示すのはそれぞれ 50Ω と 200Ω の抵抗性負荷を接続した場合の動作例です。これらの例では、ソースメータは電流 100mA を供給するようにプログラムされており、同時に電圧リミットは 10V に設定されています。

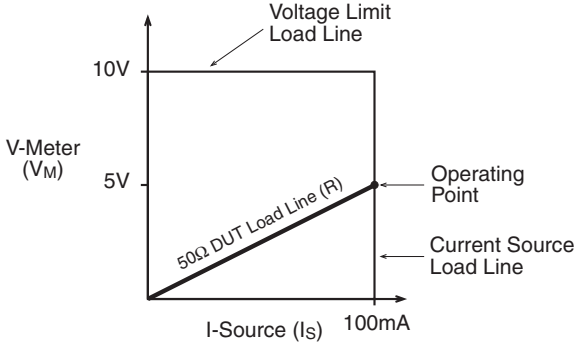
図 8-5A に示す例ではソースメータは 50Ω 負荷に 100mA を供給し、その結果として 5V が測定されます。図から分るように、50Ω 抵抗の負荷ラインは 100mA の電流ソースラインと 5V で交差します。

負荷抵抗を 200Ω まで大きくした場合の変化を示すのが図 8-5B です。200Ω の DUT 負荷ラインは電圧コンプライアンス制限ラインと途中で交わり、これによってソースメータのコンプライアンスが保たれます。このコンプライアンス条件のため、ソースメータはプログラムされた電流（100mA）を送り出すことができなくなり、200Ω DUT に対してソースメータが供給可能な電流は 50mA（10V リミットにて）までとなります。

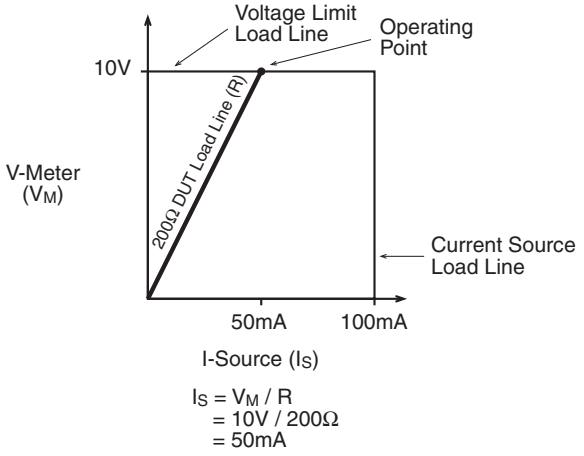
抵抗が大きくなるにつれて DUT 負荷ラインの勾配は急になってゆき、抵抗値が無限大（出力開放）に近づくとソースメータは実質的に 0mA(10V にて) しか出力できなくなります。逆に、抵抗が減少してゆくと DUT 負荷ラインの勾配が小さくなってゆき、抵抗がゼロ（出力短絡）に近づくと実質的に電圧 0V で 100mA を出力することになります。

負荷の大きさには関係なく、電圧がプログラムされたコンプライアンス値である 10V を超えることはありません。

図 8-5
電流ソース（I -Source）の動作限界



A) 通常の電流ソース動作



B) コンプライアンス条件によって制限された電流ソース

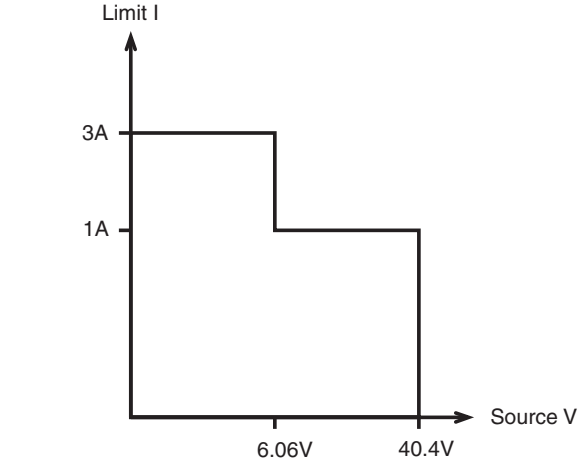
電圧ソース（V -Source）の動作限界

図 8-6 に示すのは V-Source の動作限界です。図に示されているのは第 1 象限の動作だけですが、他の 3 象限についてもほぼ同様に動作します。

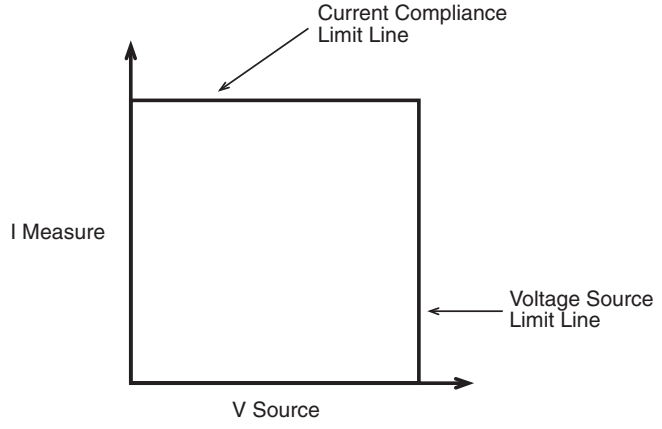
図 8-6A に示すのは V-Source の出力特性です。図から分る通り、ソースメータは 3A で 6.06V、1A で 40.4V を出力する能力を持っています。6.06V を超える電圧を供給しようとすると電流が 1A に制限されることに注意してください。

図 8-6B に示すのは V-Source の制限ラインです。電圧ソースの制限ラインは現在選択されている電圧ソースレンジで供給可能な最大ソース値を表しています。たとえば、6V ソースレンジでは 6.3V に電圧ソースの制限ラインが引かれます。電流コンプライアンス制限ラインは実効を持つ実際のコンプライアンスを表します。コンプライアンスはリアルとレンジの両方が可能であることに注意が必要です（「コンプライアンスリミット」、p.8-2、参照）。これらの制限ラインはこの動作象限におけるソースメータの動作限界を表す限界です。これらの制限ラインの内部（またはライン上）であればどこにでも動作点を置くことができます。他の象限における限界もこれと良く似ています。

図 8-6
電圧ソース（V-Source）の限界



A) 出力特性



B) 制限ライン

ソースメータが動作可能な限界は出力端子に接続されている負荷（DUT）に依存します。図 8-7 に示すのはそれぞれ 2kΩ と 800Ω の抵抗性負荷を接続した場合の動作例です。これらの例では、ソースメータは電圧 10V を供給するようにプログラムされており、同時に電流リミットは 10mA に設定されています。

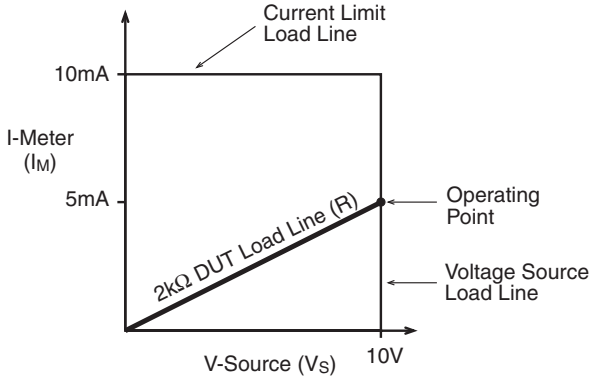
図 8-7A に示す例ではソースメータは 2kΩ 負荷に 10V を供給し、その結果として 5mA が測定されます。図から分るように、2kΩ 抵抗の負荷ラインは 10V の電圧ソースラインと 5mA で交差します。

負荷抵抗を 800Ω まで小さくした場合の変化を示すのが図 8-7B です。800Ω の DUT 負荷ラインは電流コンプライアンス制限ラインと途中で交わり、これによってソースメータのコンプライアンスが保たれます。このコンプライアンス条件のため、ソースメータはプログラムされた電圧（10V）を送り出すことができなくなり、800Ω DUT に対してソースメータが供給可能な電圧は 8V（10mA リミットにて）までとなります。

抵抗が小さくなるにつれて DUT 負荷ラインの勾配が大きくなることに注意してください。抵抗が無限大（出力端開放）に近づくと実質的に供給電流 0mA でソースメータ出力が 10V に達してしまいます。逆に、抵抗が増大してゆくと DUT 負荷ラインの勾配が小さくなってゆき、抵抗がゼロ（出力短絡）に近づくくと実質的に電流 10mA で 0V を出力することになります。

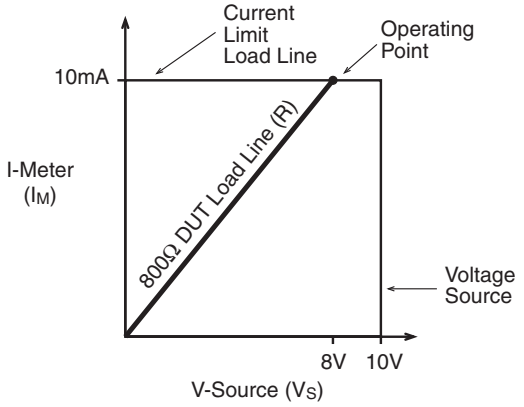
負荷の大きさには関係なく、電流がプログラムされたコンプライアンス値である 10mA を超えることはありません。

図 8-7
電圧ソース（V -Source）の動作例



$I_M = V_S / R$
$= 10V/2k\Omega$
$= 5mA$

A) 通常の電圧ソース動作



$V_S = I_M \bullet R$
$= (10mA) (800\Omega)$
$= 8V$

B) コンプライアンス条件によって制限された電圧ソース

電流ソース - 電流測定、電圧ソース - 電圧測定

ソースメータは供給しているパラメータ自体を測定することができます。すなわち、電圧を供給しながら電圧を測定することも、逆に電流を供給しながら電流を測定することも可能です。これらのメジャーソース操作での測定レンジはソースレンジと同じになります。

この機能はソースのコンプライアンスを保ちながら動作させる場合に有用です。コンプライアンス状態ではプログラムされたソース値に達しませんが、ソースを測定することによって実際の出力電圧を測定することができます。

基本回路構成

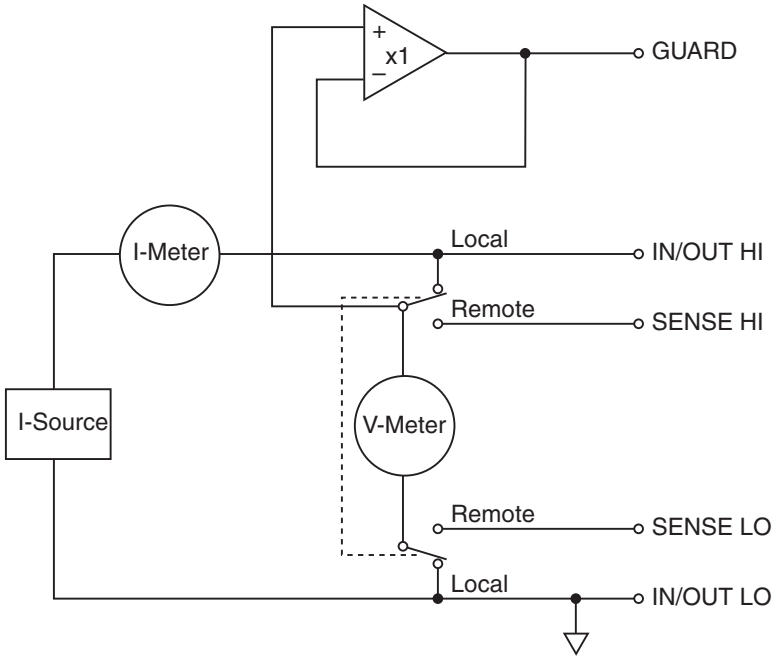
電流ソース

図 8-8 のように電流を供給（I-Source）するように構成されたソースメータは、電圧リミットを設定できる高インピーダンス電流源として機能し、かつ電流（I-Meter）と電圧（V-Meter）どちらでも測定することができます。

2 線式ローカルセンシングを使用する場合は、ソースメータの入出力端子にかかる電圧を測定します。4 線式リモートセンシングの場合は、センス端子を用いて直接 DUT 両端の電圧を測定します。これにより、試験リード線（ソースメータと DUT 間の接続）に起因する電圧降下の影響を排除した測定が可能になります。

電流ソースは電流ソース確度を上げるためにセンス導線を使用する必要はありません。4 線式のリモートセンシングを選択した場合は、センス導線を接続しなければ正確な動作は得られません。

図 8-8
電流ソース構成



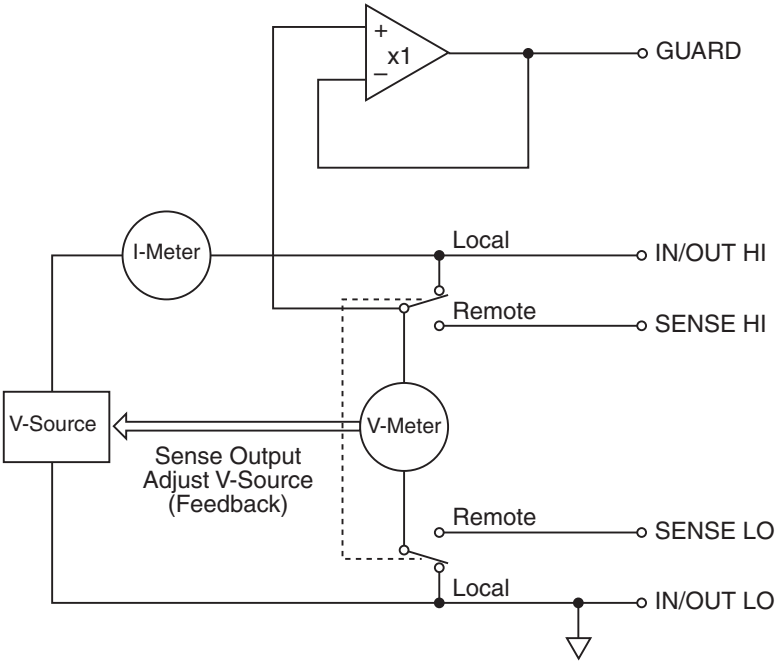
電圧ソース

図 8-9 のように電圧を供給（V-Source）するように構成されたソースメータは、電流リミットを設定できる低インピーダンス電圧源として機能し、かつ電流（I-Meter）と電圧（V-Meter）どちらでも測定することができます。

この構成では検知回路を使用して出力電圧を連続的にモニタリングし、必要に応じて電圧ソース（V-Source）の調整を行います。V-Meter は入出力端子における電圧を検知（2 線ローカルセンシング）、または DUT 両端の電圧を検知（センス端子を使用する 4 線式リモートセンシング）して、プログラムされている電圧レベルと比較します。検出されたレベルがプログラム値に一致しない場合は、その偏差に応じて V-Source を調節します。リモートセンシングを使用することにより、試験導線に起因する電圧降下の影響を排除して、プログラムされた電圧を正確に DUT 両端に印加することができます。

電圧の誤差はアナログ信号として V-Source へフィードバックされ、ソース側の誤差増幅器を使用して試験導線に発生した IR 電圧降下を補償します。

図 8-9
電圧ソース構成



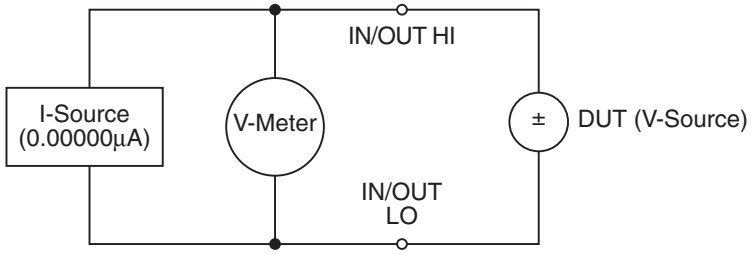
測定のみ（V または I）

ソースメータを電圧計または電流計専用として使用するための構成を図 8-10 に示します。図 8-10A に示すように、ソースを 0A にセットし、測定機能として電圧を選択することによってソースメータが電圧測定専用器になります。

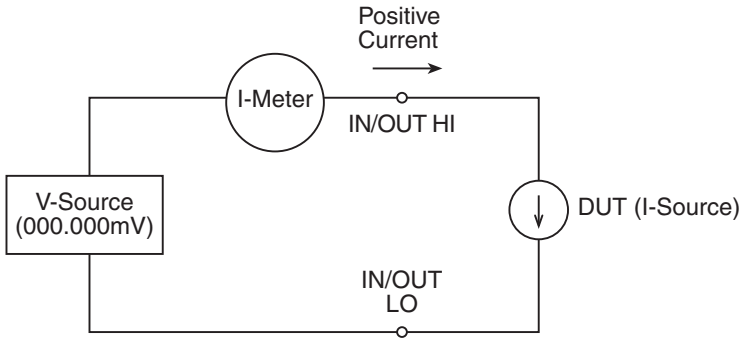
注意　**電圧コンプライアンス（V-Compliance）は被測定電圧よりも高いレベルに設定しなければなりません。この条件が守られないと過大電流がソースメータに流れて、ソースメータが破損する原因になります。また、外部電圧を I-Source に接続する場合は、出力 OFF 状態を高インピーダンスモードに設定しなければなりません。詳しくは「コンプライアンスリミット」（p.8-2）の説明をご覧ください。**

図 8-10B ではソースを 0V にセットし、測定機能として電流を選択することによってソースメータを電流測定専用器として設定しています。読み取り値が正（+）の値を示すためには通常の電流が IN/OUT の HI から LO へ流れなければならないことに注意してください。

図 8-10
測定専用の回路構成



A. 電圧測定専用



注:IN/OUT HI から正方向へ電流が流れると正(+)の測定値が得られます。

B. 電流測定専用

注:2 線式のローカルセンシングを使用してください。

ガード

警告 **ガードは出力の HI と等電位に保たれます。したがって、出力 HI に危険な高電位が出力されているときはガードにも同じ電位が印加されています。**

ガードの概要

ガード（背面パネルの GUARD 端子）には Input/Output HI（リモードセンシングの場合は Sense HI）と等電位のバッファリングされた電圧が常に出力されており、いつでも使用可能な状態にあります。ガードを使用する目的は、入出力 HI と LO 間の漏れ電流（およびキャパシタンス）の効果を打ち消すことにあります。外部試験回路の漏れ電流が大きい場合に被駆動ガードを使用しないで動作させると、ソースメータの性能に悪影響を及ぼすことがあります。

漏れ電流は次のようなシステムに寄生する漏れ電流経路はもちろん、それ以外の非寄生的な経路でも起りえます。寄生抵抗の例としてはたとえば同軸ケーブルや 3 軸ケーブルの絶縁体を通る漏れが挙げられ、非寄生的な抵抗の例としては DUT に並列に接続された抵抗を通る経路などを挙げることができます。

ガードの接続

標準的にはケーブルやテストフィクスチャのガードシールドをこのガードで駆動します。これにより、ケーブルのガードシールドを介して効果がテストフィクスチャにまで及びます。テストフィクスチャ内部では適宜、ガードプレートや DUT を囲むシールドにこのガードを接続してください。

警告 **ガードは危険な電位（>30Vrms または 42.4V peak）になることがあります。作業される方の傷害（場合によっては致命的な）を防止するために安全シールドを設けてガードに直接触れることのないようにしてください。この安全シールドはガードプレートやシールドを完全に覆う必要があり、安全設置 GND に確実に接続されていなければなりません。テストフィクスチャの金属ケースを安全シールドとして利用する例を図 8-11B に示します。**

注 ***試験装置のガード接続の詳細についてはセクション3をご覧ください。***

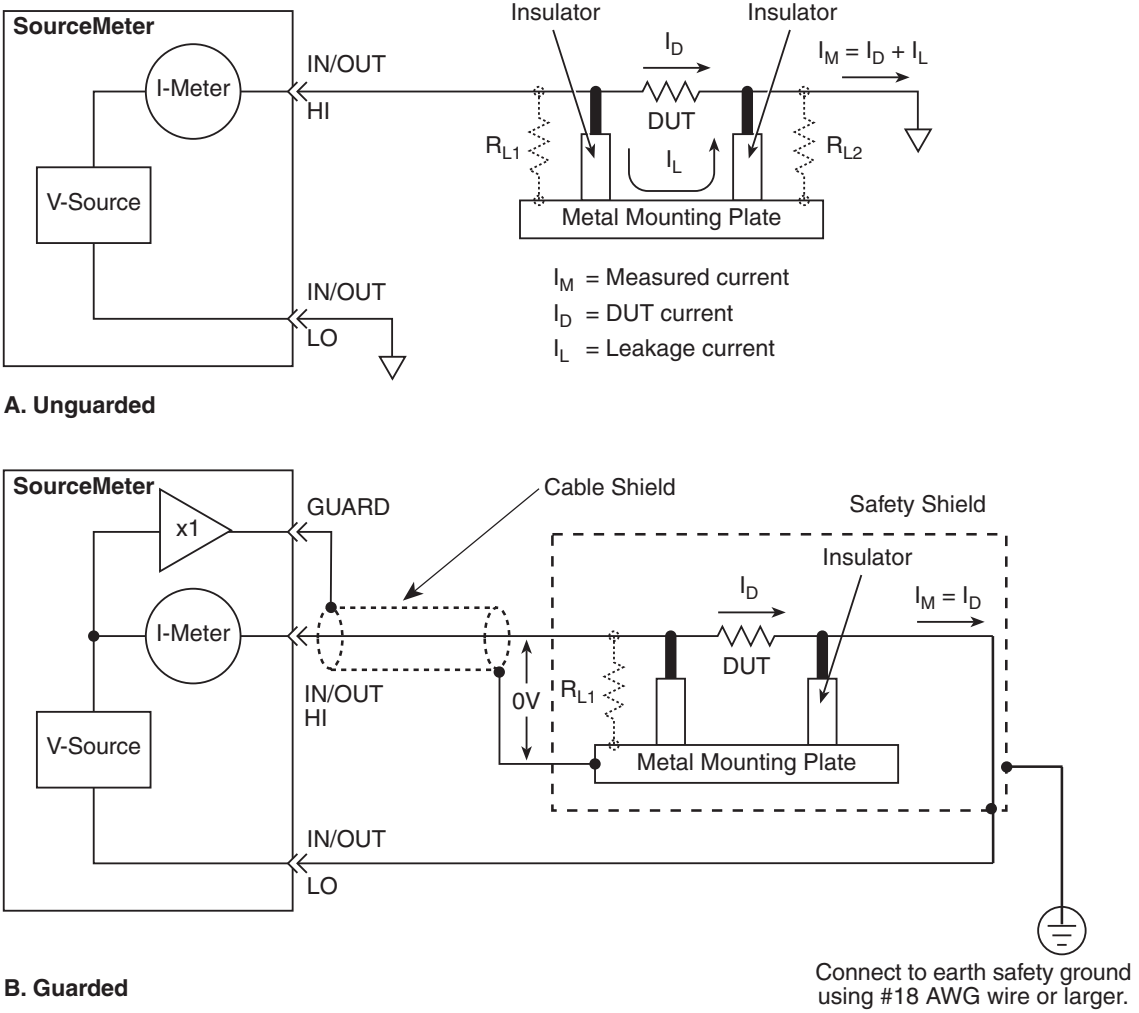
テストフィクスチャ内でガードを DUT まで延長するには 3 軸ケーブルの使用が便利です。この場合にはケーブルの中心導線を In/Out HI の接続に用い、内側のシールドをガード用として、外側のシールドを In/Out LO 用に使用して安全シールドに接続します（安全シールドはさらに安全接地 GND に接続）。

ガードの電位が 30Vrms（42.4V peak）を超えない場合には同軸ケーブルを利用することも可能です。この場合には中心の導線を In/Out HI 接続用に用い、外側シールドをガード用として利用します。これよりも高い電位が印加されるケースでは前ページで説明したように 3 軸ケーブルを使用してください。

ケーブルガードがテストフィクスチャの絶縁体を通る漏れ電流を解消する原理を示すのが図 8-11 です。図 8-11A において、漏れ電流（IL）は絶縁体（RL1、RL2）を貫流して In/Out LO に達し、DUT の微小電流（高抵抗）測定に悪影響を与えます。

一方、図 8-11B ではケーブルシールドを駆動付ガードに接続することによって、ガードを絶縁体の金属製ガードプレートまで延長しています。これにより RL1 の両端が等電位（電圧降下 0V）となることから、この抵抗を通る経路では電流が流れなくなり、したがってソースメータは DUT を流れる電流のみを測定することになります。

図 8-11
ガード使用時とガード無し時測定の比較



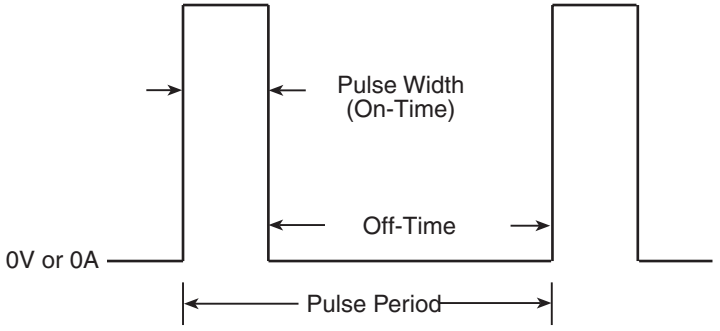
パルスの概念

ファクトリスクリプトを利用して 260x 型に固定 / 線形階段 / 対数階段パルススイープを実行させることができます（詳しくは「パルススイープ」、p.5-6、参照）。以下の項ではパルス周期、ライズ / フォールタイム、およびデューティサイクルについて詳しく解説します。ソースの過渡応答および整定時間の詳細については付録 A の仕様をご覧ください。

パルス周期

図 8-12 に示すように、パルス周期はパルスのオン時間（パルス幅）とオフ時間の合計です。パルスのオフ状態では、その時点で使用している機能に応じて 0V または 0A 出力が仮定されています。パルスがオンになると、プログラムされた電流または電圧が出力されるものとします。固定パルススイープでは全部のパルスが同じ振幅を持ちますが、階段スイープの場合にはプログラムされた値に従ってスイープステップの値を変化させてゆきます。どのタイプのパルススイープも、パルスのオン時間とオフ時間は相互に独立してプログラムすることができます。

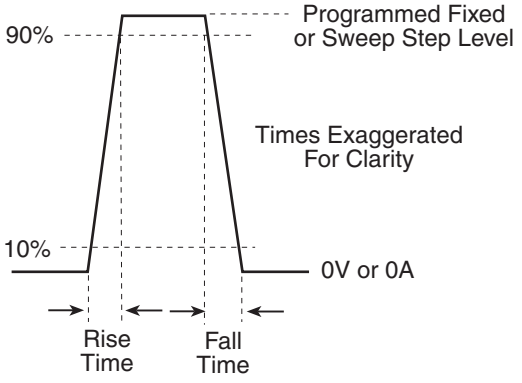
図 8-12
パルス周期



パルスのライズ / フォールタイム

図 8-13 に示すように、パルスのライズタイムとはパルスが最大値の 10% から 90 % まで上昇するのに要する時間を意味します。260x 型の場合、パルスのライズ / フォールタイムは過渡応答とソース出力の整定時間に依存し、またこれらのパラメータはどのソースレンジを選択するかによっても影響を受けます。過渡応答およびソースの整定時間について詳しくは付録 A の仕様をご覧ください。

図 8-13
パルスのライズ / フォールタイム



パルスのデューティサイクル

デューティサイクルはパルス周期の中で出力がオンになっている時間幅をパーセントで表した数値です。

デューティサイクル = パルス幅 /（パルス幅＋オフ時間）

たとえば、パルス幅が 10msec、オフ時間が 90msec の場合のデューティサイクル計算値は次のようになります。

デューティサイクル = 10msec / (10msec + 90msec)
= 10msec / 100msec
= 0.10
= 10%

9
システムの拡張 (TSP-Link)

セクション 9 のトピックス

概要 , 9-2
  マスタとスレーブ , 9-2
  システム構成 , 9-2

接続 , 9-3

初期化 , 9-3
  ノード番号の割り付け , 9-3
  TSP-Link のリセット , 9-4

拡張されたシステムの使用 , 9-6
  ノードへのアクセス , 9-6
  システムの挙動 , 9-7

概要

TSP-Link は装置と他の装置間の通信を可能にする拡張インターフェイスです。TSP-Link を使用して試験システムを拡張すると、最高 64 台までの TSP-Link 対応装置を接続することができます。

マスタとスレーブ

TSP-Link システムでは 1 つのノード（装置）がマスタとして機能し、それ以外のノードはすべてスレーブになります。

マスタはシステム内の他のノード（スレーブ）をコントロールします。いずれかのノードがローカルからリモート操作へ移行すると、そのノードがシステム内のマスタになり、それに連動して他のすべてのノードもリモート操作へ移行してスレーブになります。逆に、いずれかのリモートからローカル操作へ移行すると、他のすべてのノードもローカル操作へ移行し、ノード間に存在していたマスタ / スレーブ関係は解消します。リモートとローカル操作について更に詳しくはセクション 2 の「相違点：リモート vs. ローカル状態」をご覧ください。

スレーブはマスタによってコントロールされるノードであるため、スレーブの GPIB と RS-232 コマンドインターフェイスは使用できなくなります。

システム構成

TSP-Link システムは PC 利用システムばかりでなく PC を含まない（スタンドアローン）システムでも使用が可能です。

スタンドアローンシステム – スタンドアローンシステムでは、一台の装置の前面パネルからスクリプトを実行してシステムをコントロールします。PC が接続されている必要はありません。図 9-1 に示すシステムでは任意の一台の装置の前面パネルからスクリプトを実行することができます。

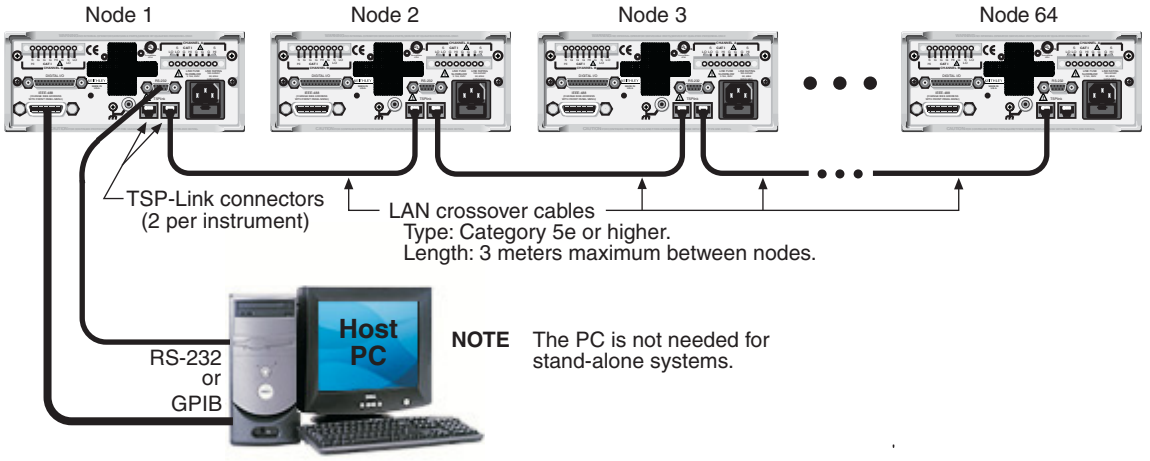
PC 利用システム – PC を利用するシステムでは、いずれか 1 つのノードへの GPIB/RS-232 による接続がシステム全体へのインターフェイスとして機能します。図 9-1 に示す例では、Node 1 に接続された GPIB/RS-232 インターフェイスがシステム全体をコントロールします。

接続

TSP-Link で拡張されたシステムの例を図 9-1 に示します。図に示すように、1 台のユニットだけが GPIB または RS-232 インターフェイスを介して PC に接続されています。PC との通信接続について詳しくはセクション 2 の説明をご覧ください。

図 9-1 に示すように、システム内のすべてのユニットは LAN クロスオーバーケーブルを使用してデイジーチェーン（芋づる式）接続されます。

図 9-1
TSP-Link 接続



初期化

TSP-Link システムは使用開始前に初期化が必要です。初期化が成功するためには、TSP-Link システム内の装置それぞれに異なるノード番号が割り付けられていなければなりません。

ノード番号の割り付け

260x 型はノード番号 1 が割り付けられて工場を出荷されます。ノード番号は不揮発性メモリに保存されますから、装置の電源が切れても失われることはありません。

前面パネル操作

TSP-Link のノード番号はそれぞれの装置の前面パネルから MAIN MENU を使用して設定することができます。具体的手順を表 9-1 に示します。

表 9-1
装置の前面パネルからノード番号を割り付ける方法

260x 型の MAIN MENU
1) MENU キーを押して MAIN MENU へアクセス
2) COMMUNICATION メニューを選択
3) TSPLINK_CFG メニューを選択
4) NODE メニューを選択
5) 装置の NODE 番号（01 - 64）を選択

リモートプログラミング

装置にノード番号を割り付けるためには属性値 tsplink.node に希望する数字を代入します。

```
tsplink.node = N
```

ここに、N = 1 ～ 64

装置のノード番号を知るには、次のコマンドを使用して属性値 tsplink.node の値を読み取ります。

```
print(tsplink.node)
```

上記の print コマンドはノード番号を出力します。たとえば、ノード番号が 1 であった場合は 1.000000e00 を出力します。

TSP-Link のリセット

ノード番号の設定がすべて終了したならば、TSP-Link をリセットしてシステムを初期化しなければなりません。初期化が成功するためには、すべてのユニットに電源が入った状態で TSP-Link をリセットする必要があります。

注 初期化後にシステムトポロジーを変更した場合にも TSP-Link をリセットしてシステムを再度初期化する必要があります。たとえば次のような変更はシステムトポロジーを変化させます。

- システム内のいずれかのユニットの電源遮断やリブート
- ユニット間の LAN ケーブルを取り外す、あるいは結線を変更する。

前面パネル操作

TSP-Link は任意の装置の前面パネルから MAIN MENU を使用してリセットすることができます。その具体的手順を表 9-2 に示します。

表 9-2
前面パネルから TSP-Link をリセットする手順

260x 型の MAIN MENU
1) MENU キーを押して MAIN MENU へアクセス
2) COMMUNICATION メニューを選択
3) TSPLINK-CFG メニューを選択
4) RESET を選択して TSP-Link のリセットを実行。

リモートプログラミング

TSP-Link のリセットに関係するコマンドをまとめて表 9-3 に示します。

表 9-3
TSP-Link リセットコマンド

コマンド	説明
`tsplink.reset()`	TSP-Link システムを初期化
`tsplink.state`	最も最近実行された TSP-Link リセットが成功であった場合は online を返し、リセット失敗であった場合は offline を返す。

以下の条件のいずれかが成立していると TSP-Link をリセットする試みが失敗します。

- システム内の 2 台以上の装置が同じノード番号を持っている。
- リセットを実行する装置に他の装置が 1 台も接続されていない。
- システム内に電源が投入されていない装置が少なくとも 1 台存在する。

プログラミング例 – 以下のコードは TSP-Link をリセットしてその状態を返します。

```
tsplink.reset()
print(tsplink.state)
```

リセットが成功すると online を返して全てのノード間の通信が確立されたことを示します。

拡張されたシステムの使用

ノードへのアクセス

TSP-Link をリセットするとノードテーブルが作られます。このテーブルにはシステム内の各ユニットに対応する項目が登録され、その項目にユニットのノード番号が付けられます。システム内のノードへのアクセスには変数 node[N]（N はノード番号）を使用します。たとえば、Node 1 はテーブル内では node[1] という項目で表されます。

これらの項目自体が 1 つのテーブルを構成しており、このテーブルにはそのユニットが共有するすべての論理装置（および付随する ICL コマンド）が記載されています（詳しくは「論理装置」、p.12-5、の説明を参照）。したがって、Node 1 の SMU A は node[1].smua としてアドレッシングが可能です。

ノードがマスタの場合は node[N] の別名変数として localnode を使用することができます（N はマスタのノード番号）。たとえば、Node 1 がマスタであった場合は node[1] の代わりとして localnode を使用することができます。

注　スレーブ上に実装されたスクリプトはリモートプログラミングを使用してアクセスすることができません。

プログラミング例 – 次に示すのは図 9-1 に示した TSP-Link システム内の装置にアクセスするためのプログラム例です。

- 次の 3 種類のコマンドのいずれかを使用して Node 1（この例ではマスタであることに注意）の SMU A をリセットすることができます。システム内の他のノードは影響を受けません。

```
smua.reset()
localnode.smua.reset()
node[1].smua.reset()
```

- 次のコマンドは Node 4（スレーブ）の SMU A をリセットします。他のノードは影響を受けません。

```
node[4].smua.reset()
```

システムの挙動

reset() コマンドの使用

TSP-Link の操作の大部分はシステム内の 1 つのノードだけを対象としますが、reset() はそれとは異なりシステム全体を対象としてコマンドです。reset() コマンドはその名の通り全てのノードをデフォルト設定の状態に戻します。

```
reset()                    -- TSP-Link システムの全ノードをリセット
```

1 つのノードだけをリセットしたい場合は node[N] または localnode を reset と組み合わせて使用します。以下の例では指定された以外のノードは影響を受けません。

```
node[1].reset()        -- Node 1 のみをリセット
localnode.reset()      -- Node 1 のみをリセット
node[4].reset()        -- Node 4 のみをリセット
```

Abort

abort は実行中のスクリプトを中断してすべてのノードをローカル操作に戻す（REM ランプ消灯）とともに、ノード間のマスタ / スレーブ関係を解除します。この中断操作を実行するにはマスタを対象として abort コマンドを発行するか、もしくはシステム内の任意のノードの EXIT キーを押します。

また、任意のノードで OUTPUT ON/OFF キーを押した場合にも同じ中断動作が行われます。動作の結果は基本的に上と同様ですが、OUTPUT ON/OFF キーを使用した場合にはさらにシステム内の全 SMU の出力が OFF になります。

10
デジタル I/O とインターロック

セクション 10 のトピックス

概要 , 10-2

デジタル I/O ポート , 10-2
ポート構成 , 10-2
デジタル I/O の回路構成 , 10-4
デジタル I/O ラインのコントロール , 10-4

インターロック , 10-8
概要 , 10-8
操作 , 10-8
コントロール , 10-9

概要

このセクションではデジタル I/O ポートの使用法を次の項目に分けて詳しく説明します。

- 「デジタル I/O ポート」、p.10-2
- 「インターロック」、p.10-8

デジタル I/O ポート

ソースメータは外部デジタル回路のコントロールに使用できるデジタル入出力ポートを備えています。たとえば、このデジタル I/O ポートでハンドラをコントロールすることによりビニング操作を実行することができます。

ポート構成

デジタル I/O ポートは装置の背面パネルに 標準 DB-25 コネクタ（メス）を使用して実装されています（図 10-1）。

接続ケーブル

デジタル I/O ポートを TLINK（Trigger Link）を備えた他の Keithley 装置へ接続するときは、オスの DB-25 コネクタ付きケーブル（Keithley 部品番号 CA- 126-1）または 2600 型 TLINK ケーブルを使用してください。

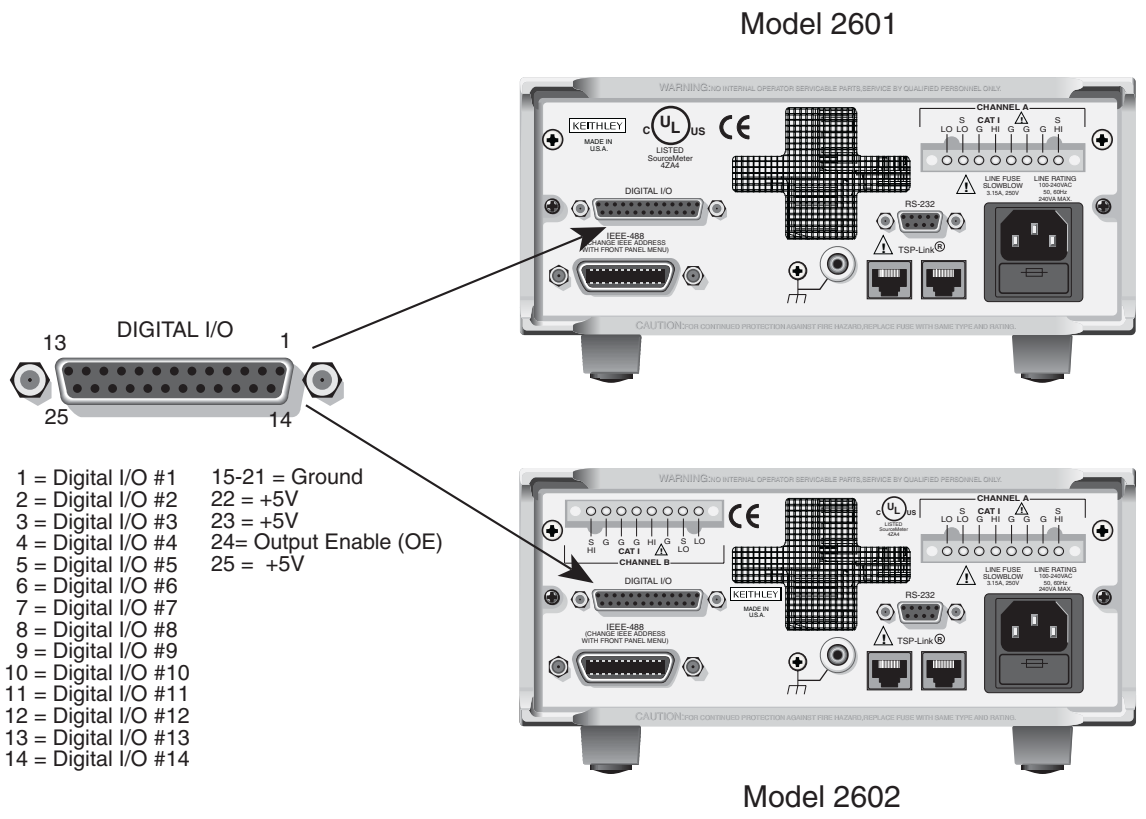
デジタル I/O ライン

ポートには 14 本のデジタル I/O ラインが出ています。各ラインは High（+5V）または Low（0V）を出力可能であるほか、High/Low 論理レベルの読み取りも可能です。

+5V 出力

デジタル I/O ポートは +5V を外部に供給するラインが出ており、これを利用して外部の論理回路を駆動することができます（このラインの最大供給電流は 600mA）。このラインは自己リセットヒューズ（1 時間で回復）で保護されています。

図 10-1
デジタル I/O ポート



インターロックライン

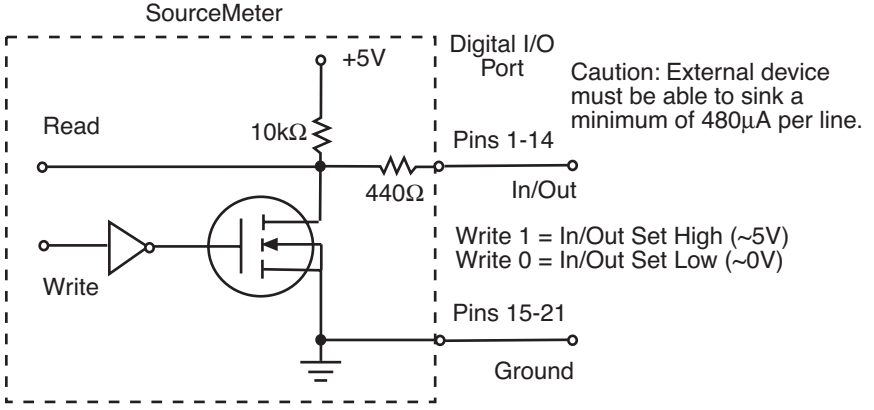
デジタル I/O のインターロック（OE: Output Enable）をテストフィクスチャや部品ハンドラのスイッチと組み合わせて使用することができます。たとえば、テストフィクスチャの蓋を開いたときに DUT の電源を強制的に遮断したい場合などにこのラインを利用するのが便利です。操作の詳細については「インターロック」（p.10-8）をご覧ください。

警告　**安全保持のためにインターロックが必要な場合は、アプリケーションの要求を満たしながらオペレータを危険な電圧から保護するための信頼できる回路を別に準備する必要があります。2600 型シリーズのデジタル I/O ポートはこのような安全回路のコントロールには適しません。安全インターロックを制御する目的でこの I/O ポートを使用しないでください。**

デジタル I/O の回路構成

デジタル I/O ポートの基本的な構成を図 10-2 に示します。ラインに 1 を書き込むと High（~ +5V）が、0 を書き込むと Low（~0V）が出力されます。外部デバイスは I/O ラインを GND へプルダウンし、デバイス側は 1 本の I/O ラインあたり少なくとも 480 μ A の電流を送り出す能力を持つ必要があります。

図 10-2
デジタル I/O ポートの回路構成



デジタル I/O ラインのコントロール

デジタル I/O ラインの主たる目的はデバイスハンドラと組み合わせて試験に制限を課すことですが、これ以外のたとえば外部論理回路のコントロールなどの目的に利用することもできます。次に示すように、I/O ラインは前面パネルとリモートのどちらからでもコントロール可能です。

デジタル I/O ビット

デジタル I/O ラインの各ビットを表 10-1 に示します。

表 10-1
デジタル I/O ビット

ライン#	ビット	10 進	16 進
1	B1	1	0x0001
2	B2	2	0x0002
3	B3	4	0x0004
4	B4	8	0x0008
5	B5	16	0x0010
6	B6	32	0x0020
7	B7	64	0x0040
8	B8	128	0x0080
9	B9	256	0x0100
10	B10	512	0x0200
11	B11	1024	0x0400
12	B12	2048	0x0800
13	B13	4096	0x1000
14	B14	8192	0x2000

前面パネルからデジタル I/O をコントロール

デジタル I/O ラインを動作させる手順 :

1. CONFIG キーを押し、続いて OUTPUT キーを押します。
2. DIO-CONTROL を選択し、ENTER または回転ノブを押します。
3. ON または OFF のいずれかを選択し、ENTER または回転ノブを押します。
4. EXIT を押して通常表示に戻ります。

デジタル I/O の出力値設定

デジタル I/O に出力値をセットする手順：

1. MENU キーを押します。
2. GENERAL を選択して、ENTER または回転ノブを押します。
3. DIGOUT を選択し、ENTER または回転ノブを押します。
4. DIG_IO_OUTPUT を選択し、続いて ENTER または回転ノブを押します。
5. デジタル I/O ラインに出力したい値を 10 進数（0 ～ 16,383：表 10-1 参照）で設定し、続いて ENTER または回転ノブを押します。
6. 通常表示に戻るには EXIT を押します。

デジタル I/O ラインの書き込み保護

特定のデジタル I/O ラインの値が変化しては困る場合には、次の手順に従って書き込み保護を加えることができます。

1. MENU キーを押します。
2. GENERAL を選択し、ENTER または回転ノブを押します。
3. DIGOUT を選択し、ENTER または回転ノブを押します。
4. WRITE_PROTECT を選択してから ENTER または回転ノブを押します。
5. 書き込み保護付きデジタル I/O ラインに出力したい値を 10 進数（0 ～ 16,383：表 10-1 参照）として設定し、続いて ENTER または回転ノブを押します。
6. 通常表示に戻るには EXIT を押します。
7. 書き込み保護を解除するときは、上と同じ作業を繰り返して同じ値を入力してください。

デジタル I/O のリモートコマンド

デジタル I/O へのアクセスとコントロールに使用するコマンドをまとめて表 10-2 に示します。これらのコマンドの完全な説明についてはセクション 12 をご覧ください。デジタル I/O ポートと個々のラインへのアクセスとコントロールに使用する 10 進 /16 進数値は表 10-1 にまとめられています。

基本デジタル I/O コマンド

これらの基本コマンドを使用して、デジタル I/O の定常状態動作（個々の I/O ラインへの書き込み / 読み取り、ポート全体への書き込み / 読み取り）をコントロールします。

デジタル I/O トリガコマンド

デジタル I/O ポートに入力される外部トリガパルスを使用して 260x 型にトリガをかける、あるいは外部デバイスに対してトリガを送信する等の操作に使用するコマンドです。

> ***注　　デジタル I/O ラインは入力と出力のどちらにでも使用できますが、入力として使用するデジタル I/O ラインすべてに 1 を書き込んでおかなければなりません。***

表 10-2
デジタル I/O コマンド

コマンド	説明
基本 I/O コマンド	
digio.readbit(bit)	入力として設定されたデジタル I/O ラインから値を読み込む
digio.readport()	デジタル I/O ポート読み込み
digio.writebit(bit, data)	出力として設定されたデジタル I/O ラインにデータを書き込む
digio.writeport(data)	デジタル I/O ポートにデータを書き込み
digio.writeprotect = mask	デジタル I/O ポートに書き込み保護マスクを設定
デジタル I/O トリガコマンド	
digio.trigger[line].assert()	デジタル I/O ラインにトリガを出力
digio.trigger[line].clear()	トリガのイベントディテクタをクリア
digio.trigger[line].mode = mode	I/O トリガイベント ディテクタのモードコントロール： digio.TRIG_FALLING (下降エッジトリガ) digio.TRIG_RISING (上昇エッジトリガ) digio.TRIG_EITHER (上昇 / 下降エッジトリガ) digio.TRIG_SYNCHRONOUS(下降エッジトリガの検出とラッチ)
digio.trigger[line].pulsewidth = width	トリガパルスの幅を設定
digio.trigger[line].release()	ラッチされているトリガを解除
digio.trigger[line].wait(timeout)	トリガ発生待ち

デジタル I/O のプログラミング例

基本デジタル I/O プログラミング例

以下のコマンドはデジタル I/O ポートのビット 1 を High に設定し、続いて全ポートの値を読み込みます。

```
digio.writebit(1,1)                --ビット 1 を High にセット
data = digio.readport()            --デジタル I/O ポート読み込み
```

デジタル I/O トリガ例

以下に示すコマンドはライン 2 のパルス幅を 10 μs、トリガモードを下降エッジに設定し、続いてデジタル I/O ライン 2 へのトリガパルス出力をアサートします。

```
digio.trigger[2].pulsewidth = 1e-5--ライン 2 のパルス幅を 10 μs に設定
digio.trigger[2].mode =            --ライン 2 を下降エッジモードに設定
digio.TRIG_FALLING
digio.trigger[2].assert()          --ライン 2 へのトリガ出力アサート
```

インターロック

概要

デジタル I/O ポートはテストフィクスチャのスイッチと組み合わせて使用することを目的としたインターロックラインを備えています。このラインをうまく使うことにより、テストフィクスチャの蓋が開いたときには自動的にソースメータ出力を OFF にすることができます。テストフィクスチャを使用する際の安全に関する重要な注意事項がセクション 3 に説明されています。

警告 **安全保持のためにインターロックが必要な場合は、アプリケーションの要求を満たしながらオペレータを危険な電圧から保護するために信頼できる回路を別に準備して頂く必要があります。2600 型シリーズのデジタル I/O ポートはこのような安全回路のコントロールには適しません。安全インターロックを制御する目的でこの I/O ポートを使用しないでください。**

操作

イネーブル状態にされたソースメータ出力は、インターロックラインがスイッチを介して +5V へプルアップされたときにだけ ON になります（図 10-3A 参照）。テストフィクスチャの蓋が開くとスイッチも開くため（図 10-3B）、インターロックラインが Low へ変化してソースメータ出力が OFF になります。逆に、インターロックを High にセットしても出力は自動的には ON にならず、その出力を High にセットしない限りは出力を再び ON にすることはできません。

コントロール

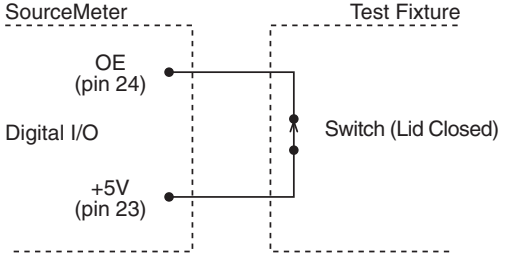
インターロックラインの動作をコントロールするには、次のコマンドを使用します。

```
smuX.source.outputenableaction = smuX.OE_NONE
smuX.source.outputenableaction = smuX.OE_OUTPUT_OFF
```

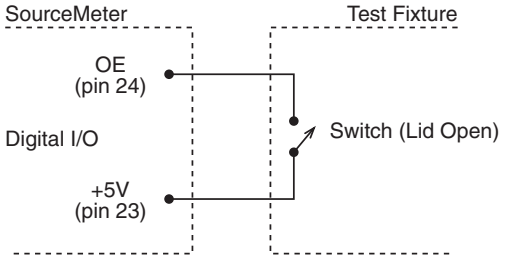
smuX.OE_NONE にセットしておくと、インターロックラインが Low へ変化してもソースメータは何の動作も行いません。smuX.OE_OUTPUT_OFF へセットしておくと、ソースメータは smuX.source.output = smuX.OUTPUT_OFF コマンドを受信したときと同様に出力を OFF にします。インターロックラインが High に戻ったとしても、ソースメータ出力は自動的には ON になりません。例えば、次のコマンドはチャンネル A のインターロックラインがオフになることを許可します。

```
smua.source.outputenableaction = smua.OE_OUTPUT_OFF
```

図 10-3
インターロックの使用法



A. OUTPUT can be turned on.



B. OUTPUT cannot be turned on.

11
通信インターフェイス

セクション 11 のトピックス

概要 , 11-2

インターフェイスの選択 , 11-2

GPIB の操作 , 11-3
GPIB 規格 , 11-3
GPIB 接続 , 11-3
一次アドレス , 11-5
ターミネータ , 11-6

一般バスコマンド , 11-7
REN (remote enable), 11-7
IFC (interface clear), 11-7
LLO (local lockout), 11-8
GTL (go to local), 11-8
DCL (device clear), 11-8
SDC (selective device clear), 11-8
GET (group execute trigger), 11-8
SPE, SPD (serial polling), 11-8

前面パネルからの GPIB 操作 , 11-9
エラー / ステータスメッセージ , 11-9
GPIB ステータス表示ランプ , 11-9
LOCAL キー , 11-10

RS-232 インターフェイスの操作 , 11-10
RS-232 インターフェイス パラメータの設定 , 11-10
データの送受信 , 11-11
ターミネータ , 11-11
ボーレート , 11-12
データビットとパリティ , 11-12
フローコントロール（信号ハンドシェーク）, 11-12
RS-232 接続 , 11-13
エラーメッセージ , 11-14

概要

このセクションでは通信インターフェイスの使用法を以下の項目に分けて詳しく説明します。

- 「インターフェイスの選択」、p.11-2
- 「GPIB の操作」、p.11-3
- 「一般バスコマンド」、p.11-7
- 「前面パネルからの GPIB 操作」、p.11-9
- 「RS-232 インターフェイスの操作」、p.11-10

注 GPIB と RS-232 通信インターフェイス自体の詳細についてはセクション 2 をご覧ください。TSP-Link も通信インターフェイスですが、こちらについてはセクション 9 に詳しく説明されています。

インターフェイスの選択

260x 型ソースメータは 2 種類の組み込みリモートインターフェイスをサポートしています。

- GPIB (General Purpose Interface Bus)
- RS-232 インターフェイス

GPIB と RS-232 インターフェイスのどちらを使用するかは、MENU キーからアクセスする COMMUNICATIONS メニューを使用して手動操作で選択するか、もしくは自動選択機能（デフォルト）を指定します。ある時点でユニットがリモート操作の対象となるのはどちらか一方のインターフェイスだけに限られます。自動選択機能を使用すると、最初にメッセージを受信した方のインターフェイスによるリモート操作の対象となり、ユニットがローカルモードに戻されるまでは他の一方のインターフェイスを完全に無視します。

いずれかのインターフェイスがすでに選択されている場合には、装置は電源が入ると同時に該当するインターフェイスの通信パラメータを表示します。自動選択がアクティブになっている場合は両方の通信パラメータを表示します。

GPIB の操作

このセクションでは GPIB の規格、バスの接続、および一次アドレスについて説明します。

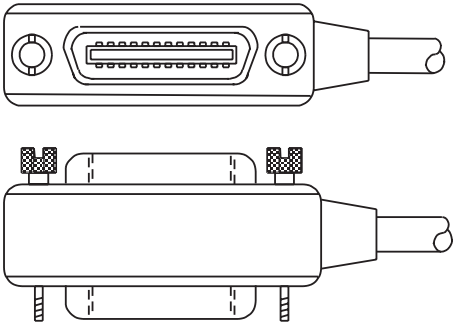
GPIB 規格

GPIB は IEEE-488 計測データバスと基本的に同じものです。この規格はハードウェアとプログラミング両面の規格を含み、IEEE（米国電気電子技術者協会）によって 1975 年に採択されました。このソースメータは IEEE-488.1 に準拠し、IEEE-488.2 の共通コマンドとステータスモデルトポロジーをサポートしています。

GPIB 接続

ソースメータを GPIB バスへ接続する際には図 11-1 に示す IEEE-488 コネクタ付きのケーブルを使用してください。

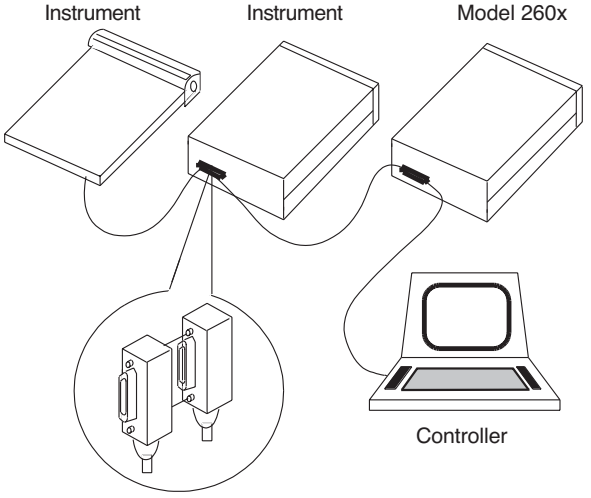
図 11-1
IEEE-488 コネクタ



1 台の装置に別な複数の装置を並列接続する場合にはコネクタを積み重ねて接続します。コネクタの両端のネジを使用してコネクタをしっかりと固定します。複数のユニットから構成される試験システムの典型的接続方式を図 11-2 に示します。

機械的な損傷防止のため、 1 台の装置に積み重ねるコネクタの数は最大 3 個までとしてください。電磁輻射による干渉を最小限に抑えるため、必ずシールド付きの IEEE-488 ケーブルを使用してください。Keithley が提供しているシールド付きケーブル一覧が「オプションとアクセサリ」（p.1-5）にまとめられています。

図 11-2
IEEE-488 接続

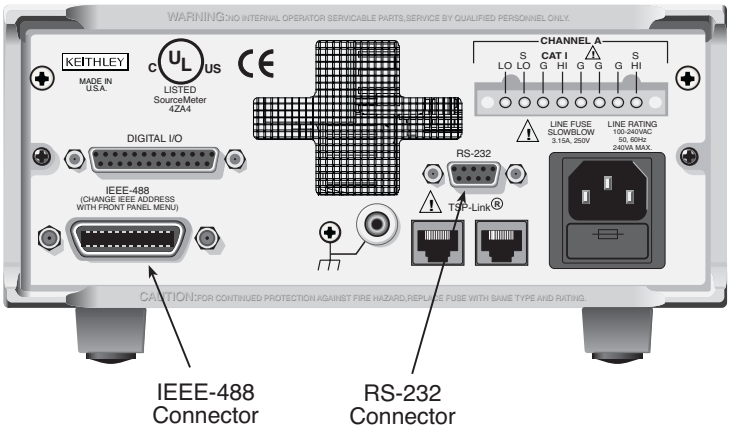


ソースメータを IEEE-488 バスへ接続するには、ケーブル側のコネクタを背面パネル上のコネクタの位置に合わせて取付け、ネジで固定します。ただし、締め付けが強くなり過ぎないように注意してください（接続位置については図 11-3 を参照）。

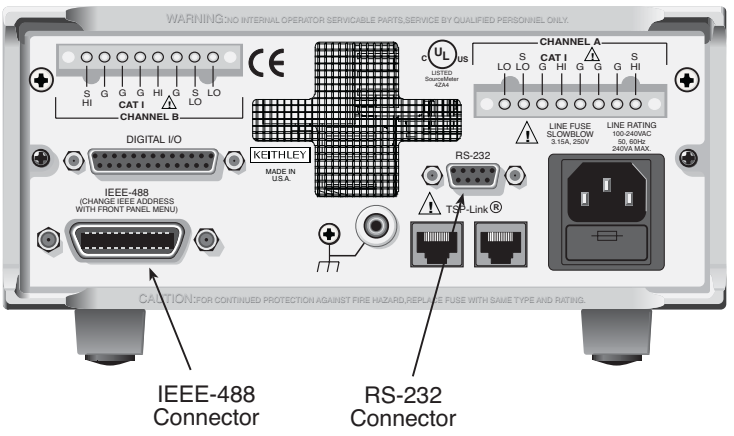
他の装置も接続する必要のあるアプリケーションの場合は同じ方法で他のコネクタも接続してください。また、ケーブルの他端が正しくコントローラに接続されていることを確認してください。IEEE-488 バスに接続できるデバイスの数はコントローラを含めて 15 台までです。許される最大ケーブル長は、20 m もしくは接続するデバイス数を 2 倍したメートル数のいずれか値の小さい方までです。これらの制限が守られない場合は正しい動作が保証されません。

図 11-3
IEEE-488 と RS-232 のコネクタ位置

Model 2601



Model 2602



一次アドレス

ソースメータは GPIB 一次アドレスが 26 に設定されて工場を出荷されます。GPIB または AUTO（自動インターフェイス選択）が選択されていると、装置に電源を入れたときに短時間だけこの一次アドレスがディスプレイに表示されます。アドレスとして 0 から 30 までの値を設定できますが、同じ GPIB バス上に存在する他のデバイスやコントローラに同一アドレスを指定することはできません（通常、コントローラアドレスとしては 0 または 21 が使用されます）。

前面パネルから一次アドレスを設定

一次アドレスを設定 / チェックするには以下の操作を行います。

1. MENU キーを押します。
2. COMMUNICATIONS を選択して、ENTER または回転ノブを押します。
3. INTERFACE_CFG を選択し、ENTER または回転ノブを押します。
4. GPIB を選択し、ENTER または回転ノブを押します。
5. 一次アドレスを希望する値に設定し、ENTER または回転ノブを押します。
6. EXIT キーを押してメニューから抜け出します。

一次アドレスのリモート設定

一次アドレスをリモート設定するには次のコマンドを使用します。

```
gpib.address = address
```

たとえば、次のコマンドはアドレスを 20 にセットします。

```
gpib.address = 20
```

GPIB アドレスの変更はこのコマンドを処理している間に有効となります。したがって、このコマンドの処理後に発生した応答メッセージはすべて新しい設定内容を反映したものになります。もし、アドレス変更メッセージ実行前に送信されたコマンドメッセージがキューに溜まっていたとすれば、以後のコマンドメッセージの途中で新しいアドレス設定が反映される可能性があります。リモートインターフェイスからアドレス属性を変更する場合にはこの点についての注意が必要です。

ターミネータ

GPIB バス経由でデータを受信すると、260x 型はラインフィード文字の受信でデータ終了と判断し、データバイトの終わりを EOI アサートで判断します（EOI アサートの伴うラインフィードも有効）。データを送信する場合にはすべての送信メッセージの最後にラインフィード文字を付加します。ターミネータであるラインフィード文字送信時に EOI ラインがアサート状態へ変化します。

一般バスコマンド

一般バスコマンド（例えば DCL）とは、装置の種類によらずに共通の意味を持つコマンドです。一般バスコマンドをまとめて表 11-1 に示します。

表 11-1
一般バスコマンド

コマンド	ソースメータへの効果
REN	次にリスナとしてアドレッシングされたときにリモート状態へ移行します。
IFC	トーカとリスナのアイドル状態に入ります。
LLO	LOCAL キーをロックアウトします。
GTL	リモート状態を解除して、前面パネルからソースメータ操作できるようにします。
DCL	すべてのデバイスを既知条件に復帰させます。
SDC	ソースメータを既知の状態へ復帰させます。
GET	トリガを起動します。
SPE, SPD	ソースメータをシリアルポールします。

REN (remote enable)

REN コマンドはコントローラによってソースメータへ送信され、装置をリモート操作可能な状態へセットアップします。一般に、バスを介して装置をプログラムする前に装置をリモートモードに入れておく必要があります。REN を True にしただけでは装置はリモート状態になりません。REN を True にしてから装置をリスナとしてアドレッシングすることによって装置がリモート状態になります。

IFC (interface clear)

IFC コマンドはコントローラがソースメータをローカル状態へ移行させるために送るコマンドであり、これによって装置はトーカ、リスナアイドル状態に入ります。装置がトーカまたはリスナに設定されていた場合は、IFC コマンドへの応答として前面パネルの TALK または LSTN ランプが消灯します。

装置へのコマンドメッセージ伝送および装置からの応答メッセージ伝送が IFC によって中断されることはありません。応答メッセージ伝送が IFC によって一時的に停止した場合は、装置がトーカとしてアドレッシングされることによって伝送が再開します。コマンドメッセージ伝送が IFC によって一時的に停止した場合は、装置がリスナとしてアドレッシングされることによって残りのメッセージが送信されます。

LLO (local lockout)

装置がリモートモードの状態に入っている間、LOCAL と OUTPUT OFF キーを除く残りすべての前面パネルからの操作は機能しません（電源スイッチはもちろん有効）。LLO コマンドを実行するとさらに LOCAL キーも機能しなくなります（OUTPUT OFF は無効にできないキーですので、LLO コマンドの影響を受けません）。

GTL (go to local)

リモートモードに入っている装置をローカルモードに戻したいときに GTL コマンドを使用します。リモート状態の解除に伴い、前面パネルのすべてのコントロール要素が機能を回復します。

DCL (device clear)

GPIB インターフェイスをクリアして既知状態に復帰する場合に DCL コマンドを使用します。DCL はアドレス指定を必要としないコマンドですから、DCL コマンドに応答できる装置すべてが同時に既知状態に復帰します。

ソースメータが DCL コマンドを受信すると、入力バッファと出力キューをクリアし、未実行コマンドをキャンセルして、さらに他のデバイスコマンドの処理を妨げているコマンドがもし存在すればそれもクリアします。DCL は装置設定や保存データには影響を与えません。

SDC (selective device clear)

SDC コマンドは実質的に DCL コマンドと同じ機能を実行しますが、アドレス指定を必要とする点が異なっています。SDC コマンドではそれぞれのデバイスを個別にアドレッシングする必要があることから、DCL コマンドがすべての装置を一斉にクリアするのに対して指定した装置のみをクリアします。

GET (group execute trigger)

GET は GPIB トリガであり、リモート操作で装置に読み取りを開始させるためのトリガとして機能します。

SPE, SPD (serial polling)

ソースメータのシリアルポールバイトを取得するにはシリアルポーリングシーケンスを実行する必要があります。このシリアルポールバイトには装置の内部機能に関する重要な情報が含まれています（付録 D 参照）。一般的に、シリアルポーリングは複数の装置のどれが SRQ ラインにサービス要求を出しているのかを確認するためコントローラが実行するシーケンスです。ソースメータからステータスバイトを取得する必要が生じた場合には、何時でも自由にシリアルポーリングシーケンスを実行することができます。

前面パネルからの GPIB 操作

このセクションでは前面パネル操作の中で GPIB のコントロールに関わる部分（メッセージやステータス表示、LOCAL キーを含む）について説明します。

エラー / ステータスメッセージ

IEEE-488 プログラミングに関連するステータス / エラーメッセージ一覧については付録 B をご覧ください。260x 型は SRQ を発生するようにプログラムすることができます。また、コマンドクエリを実行して特定のエラー条件をチェックすることも可能です。

GPIB ステータス表示ランプ

REM（リモート）、TALK（トーカ）、LSTN（リスナ）、および SRQ（サービス要求）ランプが GPIB バスの状態を表示します。それぞれのランプについて以下に説明します。

REM

REM ランプは装置がリモート状態にあることを示します。装置がリモート状態に入っている間はすべての前面パネルキー（LOCAL、OUTPUT OFF キーは例外）がロックアウトされます。REM の消灯は装置がローカル状態に戻ったことを示し、前面パネル操作も機能を回復します。

TALK

このランプは装置がトーカとしてアクティブな状態にあるときに点灯します。装置をトーカの状態に入れるには、正しい talk コマンドを使用して装置をトーカとしてアドレッシングする必要があります。ユニットがトーカとしてのアイドリング状態にあるときは TALK が消灯します。この状態に入れるには、UNT（Untalk）コマンドを送信して装置をリスナとしてアドレッシングするか、もしくは IFC（Interface Clear）コマンドを送信します。

LSTN

この表示ランプはソースメータがリスナとしてアクティブな状態になると点灯します。この状態に入れるには、正しい listen コマンドを使用して装置をリスナとしてアドレッシングする必要があります。ユニットがリスナとしてのアイドリング状態にあるときは LSTN が消灯します。この状態に入れるには、UNL（Unlisten）コマンドを送信して装置をトーカとしてアドレッシングするか、もしくはバスを介して IFC（Interface Clear）コマンドを送信します。

SRQ

エラーや特定の条件が起こったときはサービス要求（SRQ）を発生するように装置をプログラムすることができます。この表示ランプの点灯はサービス要求の発生を示します。このランプはシリアルポールバイトが読み込まれるか、または SRQ の原因となった条件がすべて解除されるまで点灯し続けます。

LOCAL キー

LOCAL（EXIT）キーは装置のリモート状態を解除してローカル操作機能を回復します。したがって、LOCAL キーを押すと REM ランプが消灯し、ユーザ定義メッセージが表示されていた場合はそれが消えて通常の表示に戻ります。

LLO（Local Lockout）コマンドが効力を持っている間は LOCAL キーも機能しません。LLO が有効である間に安全上の理由によって出力を OFF にしたい場合は OUTPUT OFF キーを使用します。LOCAL または OUTPUT OFF キーを押すと、その時点で処理中であったコマンドやスクリプトの実行も中断されることに注意してください。

RS-232 インターフェイスの操作

RS-232 インターフェイス パラメータの設定

前面パネルから操作する RS-232 パラメータ

インターフェイスパラメータを設定する手順：

1. MENU キーを押します。
2. COMMUNICATIONS を選択し、ENTER または回転ノブを押します。
3. INTERFACE_CFG を選択し、ENTER または回転ノブを押します。
4. RS-232 を選択し、ENTER または回転ノブを押します。以下の項で説明する手順にしたがって次のインターフェイスパラメータを設定します。
   - ボーレート
   - ビット数
   - パリティ
   - フローコントロール
5. EXIT キーを押してメニューから抜け出します。

リモート RS-232 パラメータ

RS-232 パラメータの設定に使用するコマンドをまとめて表 11-2 に示します。更に詳しくはセクション 12 をご覧ください。

シリアルポート設定の変更はコマンドを処理している間に有効となります。したがって、これらのコマンドの処理後に発生した応答メッセージはすべて新しい設定内容を反映したものになります。もし、設定変更メッセージ実行前に送信されたコマンドメッセージがキューに溜まっていたとすれば、以後のコマンドメッセージの途中で新しい設定内容が反映される可能性があります。リモートインターフェイスからアドレス属性を変更する場合にはこの点についての注意が必要となります。

表 11-2
RS-232 インターフェイス コマンド

コマンド	解説
serial.baud = baud	ボーレートを設定（300、600、1200、2400、4800、9600、19200、38400、57600 または 115200）
serial.databits = bits	ビット数を設定（7、または 8）
serial.flowcontrol = flow	フローコントロールを設定： serial.FLOW_NONE（フローコントロールを使用しない） serial.FLOW_HARDWARE（ハードウェアによるフローコントロール）
serial.parity = parity	パリティを設定： serial.PARITY_NONE（パリティを使用しない） serial.PARITY_EVEN（偶数パリティ） serial.PARITY_ODD（奇数パリティ）

RS-232 プログラミング例

次のコマンドを送信すると、RS-232 のボーレートが 9600、フローコントロール不使用に設定されます。

```
serial.baud = 9600
serial.flowcontrol = serial.FLOW_NONE
```

データの送受信

RS-232 インターフェイスはデータをビット数 7 または 8、1 ストップビット、パリティを偶数 / 奇数 / 不使用のいずれかで伝送します。ソースメータの接続先デバイスが同じ内容に設定されていることを確認してください。

ターミネータ

RS-232 インターフェイスを使用してデータを受信する場合、260x 型は LF を受信することによってメッセージの終了を認識します。データを送信するとき、装置はすべての送信メッセージの最後に LF を付加します。

ボーレート

ボーレートはソースメータとプログラミング端末間との通信速度を規定します。使用できる速度は次の通りです。

- 115200
- 57600
- 38400
- 19200
- 9600
- 4800
- 2400
- 1200
- 600
- 300

工場出荷時は 9600 に設定されています。

ユーザがボーレートを設定するときは、ソースメータに接続しているプログラミング用コンピュータがサポートしている速度を選択しなければならず、ソースメータと他のデバイスのボーレート設定は同一でなければなりません。

データビットとパリティ

RS-232 インターフェイスはデータ長として 7 または 8 ビット、パリティとして偶数、奇数、またはパリティ無しを選択して送受信を行います。パリティ無しを選択できるのはデータ長が 8 ビットの場合だけであることに注意してください。

フローコントロール（信号ハンドシェーク）

コントローラと装置間の通信では相手方がデータ受信を可能であるか否かを確認するために、信号のハンドシェークを行いながら信号を伝送します。

RS-232 インターフェイスはこの目的に使用する 2 本の信号ライン（RTS、CTS）を備えています（図 11-4、表 11-3 参照）。260x 型側のデータ送信準備が整うと RTS を変化させ、コンピュータからの CTS（Clear to send）受信を待って送信します。

フローコントロール の使用 / 不使用の選択は RS-232 設定メニューで行います。フローコントロールを使用するには HARDWARE、使用しないのであれば NONE を選択してください。

RS-232 接続

RS-232 シリアルポートとコンピュータ側のシリアルポートの接続には、DB-9 コネクタを取付けたストレートタイプの RS-232 ケーブルを使用します。ヌルモデムケーブルは使用できません。シリアルポートは RS-232 規格に準拠して、送信ライン（TXD）、受信ライン（RXD）、フローコントロール使用の場合は CTS と RTS、および信号 GND ラインを使用して通信を行います。装置背面ポートの RS-232 インターフェイスコネクタを図 11-4、そのピン配列を表 11-3 に示します。コネクタの取付け位置については図 11-3（p.11-5）を参照してください。

使用するコンピュータの RS-232 インターフェイスが DB-25 コネクタを使用している場合は一方の端に DB-25、他端に DB-9 コネクタを備えたケーブルを使用するか、またはアダプタが必要になります。Keithley が提供している RS-232 ケーブル一覧が「オプションとアクセサリ」（p.1-5）にまとめられています。

図 11-4
RS-232 インターフェイス コネクタ

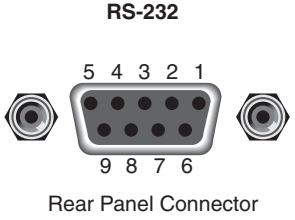


表 11-3
RS-232 コネクタのピン配列

ピン番号	説明
1	未使用
2	TXD、データ送信
3	RXD、データ受信
4	未使用
5	GND、信号 GND
6	未使用
7	RTS、送信要求
8	CTS、送信可
9	未使用

コンピュータ（PC）側の 9 ピンコネクタ（DB-9）、25 ピンコネクタ（DB-25）の信号配列を次の表に示します。

表 11-4
PC のシリアルポート ピン配列

信号 *	DB-9 のピン番号	DB-25 のピン番号
DCD、データキャリア検出	1	8
RXD、受信データ	2	3
TXD、送信データ	3	2
DTR、データ端末レディ	4	20
GND、信号 GND	5	7
DSR、データセットレディ	6	6
RTS、送信要求	7	4
CTS、送信可	8	5
RI、リング表示	9	22

* 260x 型は RS-232 信号セットの一部のみを使用します。表 11-3 参照。

エラーメッセージ

RS-232 に関連するエラーメッセージ一覧が付録 B に掲載されています。

12
装置コントロールライブラリ

セクション 12 のトピックス

コマンドのプログラミングに関する注意事項 , 12-2
- 規約 , 12-2
- 関数と属性値 , 12-3
- TSP-Link のノード , 12-5
- 論理装置 , 12-5
- 読み込みバッファ , 12-6
- 時間と日付の値 , 12-8

ICL 関数と属性値一覧 , 12-9

beeper 関数と属性値 , 12-11

bit 関数 , 12-12

delay 関数 , 12-18

digio 関数と属性値 , 12-19

display 関数と属性値 , 12-24

errorqueue 関数と属性値 , 12-40

exit 関数 , 12-42

format 属性値 , 12-42

gpib 属性値 , 12-45

localnode 属性値 , 12-46

makegetter 関数 , 12-48

opc 関数 , 12-50

printbuffer 関数と printnumber 関数 , 12-51

reset 関数 , 12-53

serial 関数と属性値 , 12-53

setup 関数と属性値 , 12-56

smuX 関数と属性値 , 12-57

timer 関数 , 12-86

trigger 関数 , 12-87

tsplink 関数と属性値 , 12-88

userstring 関数 , 12-89

waitcomplete 関数 , 12-91

コマンドのプログラミングに関する注意事項

規約

コマンドの説明では次のような規約を用います。これらの規約を理解して先へ進んでください。

ワイルドキャラクタ

多くの SMU コマンドの説明ではワイルドキャラクタを使用して一般化した形式が用いられます。このような場合のワイルドキャラクタは SMU チャンネルや機能、トリガラインを一般化して表します。ワイルドキャラクタは説明を一般化するために便宜的に用いるものであり、実際に装置に送信されるコマンドには含まれないことに注意してください。

X と Y

関数と属性に現われる X は SMU チャンネル（a または b）を表し、Y は SMU の機能（v、I、r または p）を表すために用いられます。たとえば、ソースの出力設定のための属性値は次のように一般化して表現されます。

```
smuX.source.levelY
```

SMU のチャンネル A を 5V にプログラムしたい場合は、次のコマンド文を装置へ送ります。

```
smua.source.levelv = 5.0
```

SMU のチャンネル B を 1mA にプログラムしたい場合は、次のコマンド文を装置へ送ります。

```
smub.source.leveli = 0.001
```

ワイルドキャラクタ（X,Y）は表記を一般化するためにコマンドの説明のみで使用されるものであり、ワイルドキャラクタそのものが装置に送られることは決してありません。

[N]

関数または属性値に現われる角カッコ（[ ]）で囲まれた N はデジタル I/O ラインの番号（1 - 14）を表します。たとえば、出力トリガをアサートする関数は一般に次のように表現されます。

```
digio.trigger[N].assert
```

トリガライン 5 の出力トリガをアサートするように 260x 型をプログラムしたいのであれば、次のコマンドを装置に送ります。

```
digio.trigger[5].assert()
```

ワイルドキャラクタ N を装置に送るのではないことに注意してください。ただし、角カッコ（[ ]）はコマンドから省略できません。また、上のコマンド例では丸カッコ（()）を最後に付加する必要があることにも注意してください（「関数」、p.12-3、参照）。

関数と属性値

コマンドには関数をベースとするものと、属性値をベースをするものがあります。

関数

関数をベースとするコマンドは装置の動作や活動をコントロールします。たとえば、電圧測定は SMU の動作（活動）の例です。関数をベースとするコマンドすべてが 260x 型の操作に直接関係しているとは限りません。たとえば、bit.bitand 関数は 2 つの数字の論理和（AND）をとります。

それぞれの関数は関数の呼び名に丸カッコ（()）を付加して構成されます。パラメータを必要としない関数の場合、このカッコ内は次の例のように空白になります。

`digio.writeport(15)`	デジタル I/O ライン 1、2、3、4 を High にセットする
`digio.writebit(3, 0)`	ライン 3 を Low（0）にセットする
`smua.reset()`	SMU A をデフォルト設定の状態に戻す
`digio.readport()`	デジタル I/O ポート読み込み

関数呼び出しの結果を使用するには、関数の返り値を適当な変数に代入してその変数をアクセスします。次に示すのは SMU A の電圧を測定してその値を変数 reading に返すコード例です。

```
reading = smua.measure.v()
print(reading)
```

Output: `2.360000e+00`

上に示した出力（Output）は電圧読み取り値が 2.36V であったことを示しています。

1 つの値だけを返す関数の場合は、その関数呼び出しを文の中で制御に利用することも可能です。例えば：

```
if smua.measure.v() > 5 then
    ...
end
```

属性値

属性値とは装置の機能や操作を表す特性値です。たとえば、ソース機能、レンジ、出力レベルなどが SMU ソースの特性に含まれます。

属性値への値の割り付け

属性値に新しい値を割り付けるには属性値ベースのコマンドを使用します。多くの属性値では、この値は具体的な数値もしくは予め定義されて識別子で表現されます。たとえば、属性値の 1 つであるフィルタタイプに移動平均を指定するのであれば、属性値に次のいずれかの値を割り付けます。

0、または smuX.FILTER_MOVING_AVG.
具体的には次のいずれかのコマンドメッセージによって移動平均フィルタが SMU A に設定されます。

```
smua.measure.filter.type = 0
smua.measure.filter.type = smua.FILTER_MOVING_AVG
```

属性値の幾つかは、有効範囲内であれば任意の数値を受け付けます。たとえば、電圧ソースに設定可能な電圧値の範囲は -40.4V ～ +40.4V までです。次のコマンドメッセージを送ることにより SMU A ソースのレベルは 1.53V に設定されます。

```
smua.source.levelv = 1.53
```

属性値の読み取り

属性値を読み取るには、関数にその属性をパラメータとして与えて実行するか、または別な変数に属性値を割り付けます。

パラメータ引渡しの例 – 次に示すコマンド例では属性値である SMU A のフィルタタイプを print 関数のパラメータとして与え、その値を出力させています。

```
print(smua.measure.filter.type)
```

Output: `0.000000e+00`

上の出力は、移動平均フィルタが設定されていることを示します。

変数へ割り付ける例 – 次のコマンド例は属性値を filtertype という変数に代入してフィルタタイプを読み取ります。

```
filtertype = smua.measure.filter.type
```

構文規則

- 関数や属性値を使用するコマンドは大文字と小文字を区別します。原則として、すべての関数名と属性値名は小文字で表現しなければなりませんが、パラメータは大文字と小文字の混在を許します。ただし、定数として使用する属性値名はすべて大文字で構成しなければなりません。例：

  ```
  smua.source.func = smua.OUTPUT_DCVOLTS
  ```

  電圧ソース機能を選択するための上のコマンドに現われる OUTPUT_DCVOLTS は属性定数です。
- 関数は空白文字を含んでいてもかまいません。次に示すのはデジタル I/O ライン 3 を Low に設定する関数例ですが、空白文字を含んでいてもいなくても動作は同じです。

  ```
  digio.writebit(3,0)          -- 文字列に空白文字を含まない。
  digio.writebit (3, 0)        -- 文字列が空白文字を使用している。
  ```
- コマンドの中には複数のパラメータを必要とするものがあります。複数のパラメータを記述するときは、上記の digio.writebit 関数のようにパラメータをコンマ（,）で区切ってください。

TSP-Link のノード

TSP-Link バスに接続された個々の装置やエンクロージャは必ず一意に識別可能でなければなりません。識別用の値を TSP-Link のノード番号と呼び、そのエンクロージャをノードと呼びます。それぞれのノードに他とは重複しない一意のノード番号を与える必要があります。

TSP から見ると、ノードはテーブル（表）のようなものです。ノードと呼ばれる大域定義テーブルが 1 つ存在して、その中に実際のノードがすべて記述され、それぞれのノード自体が 1 つのテーブルを構成しています。個々のノードをアクセスするには node[N]（N はそのノードに割り付けられたノード番号）を使用します。それぞれのノードは、付随するテーブル要素としてアクセス可能な幾つかの属性値を備えています。これらの属性値は次の通りです。

id　　ノードに割り付けられたノード番号
model　　ノードに対応する製品モデル名を表す文字列
revision　　ノードに対応する製品のバージョン番号を表す文字列
serialno　　ノードに対応する製品のシリアル番号を表す文字列
その他、ノードに対応する論理装置を表す項目も含まれています（「論理装置」の項参照）。
ノードを実行しているノードを表すエイリアスとして localnode という変数が用意されていますから、スクリプトを実行しているノードのノード番号を個別に調べる必要はありません。たとえば、ノード 5 上でスクリプトが実行されている場合、グローバル変数 localnode は node[5] のエイリアスとして機能します。

論理装置

通常、ユーザは 1 つのノードまたはエンクロージャに含まれる機能の総体を 1 つの装置と考えます。しかし、TSP/ICL の観点からは、個々の SMU を 1 つの装置と考えるのが便利です。説明の混乱を避けるため、エンクロージャ内の総体を機能に則して分割した SMU や部分機能を「論理装置」と呼ぶことにします。

個々の論理装置にもシステム内で一意に定まる識別子が与えられます。論理装置をコントロールする ICL 関数呼び出しにもこれらの識別子を使用します。260x 型の SMU はエンクロージャごとに以下の論理装置を備えています。

beeper	errorqueue	smua	status
digio	gpib	smub	tsplink
display	serial	timer	

論理装置も TSP からはテーブルのように見えます。それぞれの論理装置に固有な属性値と論理装置が応答するコマンド群に加えて、次に列挙する数種類の属性値は論理装置に関する情報を提供してくれます。

name　　論理装置名を表す文字列。例：`smua`.
node　　その論理装置が部分として属する TSP-Link ノードへの参照。

それぞれの論理装置は自分がサポートするコマンドごとに 1 つの要素を持っており、このセクションではこれらのコマンドについて解説します。smua と smub は同じコマンドセットをサポートしますから、両者をまとめて smuX と表記します。

ノードに属する論理装置識別子は、そのノード内においてグローバル変数として扱われます。すなわち、一般にはノードの要素としてアクセスしますが、そのノード上で実行するスクリプトではノード修飾子を使用せずに直接アクセスすることができます。たとえば、node[5] の smua に measure.v コマンドを実行させるには node[5].smua.measure.v() という表現を使用しますが、そのコマンドが node[5] 上で発行（実行）されるのであれば smua.measure.v() で充分です。TSP-Link を介して複数のユニットをコントロールする場合にはノード番号を省略しないように注意してください。

読み込みバッファ

読み取り値取得のために何通りかの方法が用意されていますが、基本的には同期式またはオーバーラップ方式で読み取ります。さらに、1 点測定用ルーチンの設定を変えて多点測定用とし、通常は 1 点測定のみを行うという方法も可能です。読み取りで得られる値は測定値のみではないことにも注意が必要です。すなわち、測定値の読み取りには測定ステータス（“In Compliance”または “Over ranged”）も付随します。

測定値を返すすべてのルーチンはこれらの値を読み込みバッファとして返します。オーバーラップ測定の場合は常に測定値が読み込みバッファに書き込まれて返されます。同期測定の場合は単一の値を返すか、もしくは単一の値と読み込みバッファ両方を返します。使用法に馴れてくれば、読み込みバッファに書かれた追加情報にアクセスして活用できるようになります。

読み込みバッファは TSL テーブルに基いて構成されており、測定値自体は通常の配列と同じ方法でアクセスすることができます。読み込みバッファが rb であったとすれば、最初の測定値は rb[1]、9 番目の測定値は rb[9] としてアクセスします。テーブル内の追加情報はそのテーブルの追加メンバーとしてアクセスします。以下に列挙する値は読み込みバッファごとにアクセスが可能です（例：rb.appendmode）。

`appendmode`	Off（デフォルト）または on。Off の場合、新しい測定の内容がそれ以前の内容を上書きしてバッファに書き込まれます。On の場合は、新しい測定の最初の内容がそれまで rb[n+1] であった位置に書き込まれます。
`basetimestamp`	rb[1] に読み取り値を書き込んだ時刻（秒単位）の、装置への電源投入を起点としたタイムスタンプです。
`capacity`	読み取りバッファに保存可能な読み取り値の総数。
`collectsourcevalues`	このモードを ON にすると、ソース値が読み取り値といっしょにバッファに保存されます。このため、1 個の読み取りあたり 4 バイトが余分に必要となります。この要素は ON または OFF（デフォルト）のいずれかの値をとり、変更できるのはバッファが空のときだけです。

collecttimestamps	この値を ON にすると、タイムスタンプが読み取り値といっしょにバッファに保存されます。このため、1 回の読み取りあたり 4 バイトが余分に必要となります。このパラメータは ON または OFF（デフォルト）のいずれかの値をとり、変更できるのはバッファが空のときだけです。
n	読み取りバッファに書き込まれている読み取り値の数。
timestampresolution	タイムスタンプの時間分解能（秒単位）を示します。デフォルト分解能は 0.000001 秒に設定されています。この分解能を指定すると、バッファに最高 71 分までの相互に重複しないタイムスタンプを保存することができます。長時間を要する試験の場合はこの値を大きくしてください。

以下の値は読み取りを実行するたびに更新されます。たとえば rb.measurefunctions[3] がイネーブルされているとき。それぞれは実際にはネスティングしたテーブルであり、関連項目は対応する測定と同じインデックスの場所に保存されます。

measurefunctions	測定値読み取りの対象となった機能（電流 / 電圧 / 抵抗 / ワット）を示す文字列配列（TSL テーブル）。
measureranges	測定実施時に使用した測定レンジのフルスケール値を保存した配列（TSL テーブル）。
readings	読み取り値バッファに保存されている読み取り値を保存した配列（TSL テーブル）。この配列は読み取りバッファを直接アクセスしたのと同じデータを保持します。たとえば、rb[2] と rb.readings[2] は同じ値です。
sourcefunctions	測定実行時のソース機能を示す文字列（電流 または 電圧）を保存した配列（TSL テーブル）。
sourceoutputstates	ソースの状態を示す文字列（Off または On）を保存した配列（TSL テーブル）。
sourceranges	測定実施時に使用したソースレンジのフルスケール値を保存した配列（TSL テーブル）。
sourcevalues	ソース機能をオン（上の collectsourcevalues の説明参照）にしておくと、読み取り時に機能していたソース値がこの配列（TSL テーブル）に保存されます。
statuses	バッファ内のすべての読み取り値のステータス値を保存した配列（TSL テーブル）。ステータス値を浮動少数点にエンコードした値が書き込まれます。
timestamps	読み取り実行時のタイムスタンプ（秒単位）を保存した配列（TSL テーブル）。この配列の値をバッファのベースタイムスタンプと比較することにより経過時間を知ることができます。

たとえば、読み込みバッファ内の読み取り値の数を知りたい場合は rb.capacity を使用します。

時間と日付の値

日付 / 時間の値はベースとなるある時間を起点として秒単位で表現されます。ベースとなる時間は次の 3 種類のいずれかです。

1. UTC（世界協定時間）12:00 am Jan 1 1970
2. 260x 型に電源を入れた時刻
3. あるイベント（たとえば、最初の読み取り値を読み取りバッファに保存した時刻）が発生した時刻

世界協定時間（UTC）の 1970 年 1 月 1 日 12:00 am を起点とする秒数で時間を表すことができます。この時間形式をコマンドで使用するためには、たとえば校正日付などを絶対時間の基準として与える必要があります。

装置に電源を入れた時刻からの経過秒数として時間を表現することも可能です。os.clock() 関数はこの形式で時間を返します。

時間を秒数で表現する形式を「標準時間形式」と呼びます。ここで注意が必要なのは、数値が浮動小数点数として表現されるため、経過時間が基準ベースから離れるにしたがって日付 / 時間の精度が劣化してゆくことです。精度の低下は全経過時間に対して 0.06ppm 程度の値になります。この精度は一般に装置の時間ベースよりもはるかに高く、その誤差が問題を引き起こすことはありませんが、ユーザが殆ど気付かないタイムベースのドリフトと直接比較することになるため相違が明瞭に現われてしまうことがある点に注意が必要です。

ICL 関数と属性値一覧

beeper 関数と属性値、p.12-11			
beeper.beep	beeper.enable		

bit 関数、p.12-12			
bit.bitand	bit.clear	bit.set	bit.toggle
bit.bitor	bit.get	bit.setfield	
bit.bitxor	bit.getfield	bit.test	

delay 関数、p.12-18
delay

digio 関数と属性値、p.12-19			
digio.readbit	digio.trigger[N].clear	digio.trigger[N].release	digio.writeport
digio.readport	digio.trigger[N].mode	digio.trigger[N].wait	digio.writeprotect
digio.trigger[N].assert	digio.trigger[N].pulsewidth	digio.writebit	

display 関数と属性値、p.12-24			
display.clear	display.inputvalue	display.prompt	display.smuX.digits
display.getannunciators	display.loadmenu.add	display.screen	display.smuX.measure.func
display.getcursor	display.loadmenu.delete	display.sendkey	display.trigger.clear
display.getlastkey	display.locallockout	display.locallockout	display.trigger.wait
display.gettext	display.menu	display.settext	display.waitkey

errorqueue 関数と属性値、p.12-40		
errorqueue.clear	errorqueue.count	errorqueue.next

exit 関数、p.12-42
exit

format 属性値、p.12-42		
format.asciiprecision	format.byteorder	format.data

gpib 属性値、p.12-45
gpib.address

localnode 属性値、p.12-46		
localnode.linefreq	localnode.prompts	localnode.showerrors

makegetter 関数、p.12-48	
makegetter	makesetter

opc 関数、p.12-50
opc

printbuffer 関数と printnumber 関数、p.12-51

printbuffer	printnumber

reset 関数、p.12-53

reset

serial 関数と属性値、p.12-53

serial.baud	serial.flowcontrol	serial.read
serial.databits	serial.parity	serial.write

setup 関数と属性値、p.12-56

setup.poweron	setup.recall	setup.save

smuX 関数と属性値、p.12-57

smuX.cal.date	smuX.measure.filter.type	smuX.nvbufferY.collecttimestamps
smuX.cal.due	smuX.measure.interval	smuX.nvbufferY.n
smuX.cal.lock	smuX.measure.lowrangeY	smuX.nvbufferY.timestampresolution
smuX.cal.password	smuX.measure.nplc	smuX.reset
smuX.cal.polarity	smuX.measure.overlappedY	smuX.sense
smuX.cal.restore	smuX.measure.rangeY	smuX.source.autorangeY
smuX.cal.save	smuX.measure.rel.enableY	smuX.source.calibrateY
smuX.cal.state	smuX.measure.rel.levelY	smuX.source.compliance
smuX.makebuffer	smuX.measure.Y	smuX.source.func
smuX.makebuffer	smuX.measureYandstep	smuX.source.levelY
smuX.measure.autorangeY	smuX.nvbufferY	smuX.source.limitY
smuX.measure.autozero	smuX.nvbufferY.appendmode	smuX.source.lowrangeY
smuX.measure.calibrateY	smuX.nvbufferY.basetimestamp	smuX.source.offmode
smuX.measure.count	smuX.nvbufferY.capacity	smuX.source.output
smuX.measure.filter.count	smuX.nvbufferY.clear	smuX.source.outputenableaction
smuX.measure.filter.enable	smuX.nvbufferY.collectcourcevalues	smuX.source.rangeY

status 関数と属性値について付録 D に説明があります。

timer 関数、p.12-86

timer.measure.t	timer.reset

trigger 関数、p.12-87

trigger.clear	trigger.wait

tsplink 関数と属性値、p.12-88

tsplink.node	tsplink.reset	tsplink.state

userstring 関数、p.12-89

userstring.add	userstring.catalog	userstring.delete	userstring.get

waitcomplete 関数、p.12-91

waitcomplete

beeper 関数と属性値

beeper 関数はブザーを鳴らします。試験やその他の開始 / 完了を外部に通知することに通常使用されます。

beeper.beep	
機能	ブザー音を発生する。
使用法	`beeper.beep(duration, frequency)` `duration` 0.1 ～ 100（秒）までの値を設定 `frequency` 453、621、987、2400（Hz）を設定
備考	• ブザーの鳴動周波数は 453Hz、621Hz、987Hz、2400Hz のいずれかです。それ以外の周波数が指定された場合は、その値に一番近い周波数が選択されます。 • ブザーの鳴動が許可されていない場合（beeper.enable 属性参照）は beeper を実行してもブザーが鳴りません。 • この関数はオーバーラップコマンドですから、スクリプトはブザー鳴動の終了を待つことなく次の処理へと進みます。前回のブザーが鳴り終わる前に次の beep を実行すると、前回の鳴動は強制打ち切りになります。beep 終了まで次のコマンドの実行開始を待たせたい場合は waitcomplete 関数を使用してください。
参照	beeper.enable

例	ブザーの動作を許可してから、2400Hz で 2 秒間ブザーを鳴らします。
	`beeper.enable = 1`
	`beeper.beep(2, 2400)`

beeper.enable	
属性値	ブザーコントロール（ON/OFF）
使用法	`beeperstate = beeper.enable` -- ブザーの状態読み取り `beeper.enable = beeperstate` -- ブザーの状態設定 ブザーを次のいずれかの状態に設定します。 `0` ブザー鳴動禁止 `1` ブザー鳴動許可
備考	• ブザーの鳴動を許可 / 禁止します。ブザーの鳴動を禁止すると、前面パネルキーのクリック音もしなくなります。 • 電源を再投入するとブザーの鳴動は許可状態になります。reset 関数はブザーの状態に影響を与えません。
参照	beeper.beep

例	ブザーの動作を許可してから、2400Hz で 2 秒間ブザーを鳴らします。
	`beeper.enable = 1`
	`beeper.beep(2, 2400)`

bit 関数

論理演算とビット演算

bit 関数は 2 つの与えられた数値に対するビット単位での論理演算、および 1 つの与えられた数値に対するビット演算を行います。与えられた数値の少数部分を切り捨てて整数化してから論理 / ビット演算を行います。

注　TSP 内部ではすべての数値を単精度の IEEE-754 浮動小数点数として保存します。数値の内部表現として保存されるのは数値データの 24 ビットだけです。論理演算は 0 から 4294967295 までのすべての整数値に対して正しく処理を行いますが、返り値として保存されるのは上位 24 ビットだけです。

論理演算 – このグループの bit.bitand、bit.bitor および bit.bitxor 関数は 2 つの数値を対象とする論理演算を行います。TSP は与えられた 2 つの整数値に対応する 2 進表現のビット列に対してビットごとに指定された論理演算を実行します。すなわち、最初の数値の第 1 ビットと 2 番目の数値の第 1 ビットの AND/OR/XOR をとり、最初の数値の第 2 ビットと 2 番目の数値の第 2 ビットの AND/OR/XOR をとる、という具合に両方の数値の全部の対応するビット間でビット単位の論理演算を行います。論理演算の結果は整数値として返されます。

ビット演算 – このグループの残りの関数は与えられた数値のビットに対する演算を行います。具体的にはビットのクリア、ビット反転、ビットのチェック、ビット（またはビットフィールド）セット、ビットの重み値（またはフィールド値）を読み取るなどの操作を行います。これらの関数はすべて、与えられた数値のビット位置を指定するインデックスパラメータを必要とします。与えられた数値の最下位ビットのインデックスが 1、最上位ビットのインデックスが 32 になります。

bit.bitand	
機能	2 つの数値間でビット単位の論理積（AND）を計算します。
使用法	`value = bit.bitand(value1, value2)` `value1` AND 演算の対象となる第 1 引数 `value2` AND 演算の対象となる第 2 引数 `value` AND 演算の返り値
備考	• この関数は 2 つの数値間でビット単位の論理積（AND）を計算します。 • value1 と value2 の少数部を切り捨てて整数値として扱います。返り値も整数です。 • さらに詳しくは「論理演算とビット演算」（p.12-12）の説明をご覧ください。
参照	bit.bitor, bit.bitxor

例	10 進数 10（2 進数 1010）と 10 進数 9（2 進数 1001）の論理積（AND）をとると 10 進数 8（2 進数 1000）が得られます。
	`value = bit.bitand(10, 9)`
	`print(value)`
	Output: `8.000000e+00`

bit.bitor	
機能	2 つの数値間でビット単位の論理和（OR）を計算します。
使用法	`value = bit.bitor(value1, value2)` `value1` OR 演算の対象となる第 1 引数 `value2` OR 演算の対象となる第 2 引数 `value` OR 演算の返り値
備考	• この関数は 2 つの数値間でビット単位の論理和（OR）を計算します。 • value1 と value2 の少数部を切り捨てて整数値として扱います。返り値も整数です。 • さらに詳しくは「論理演算とビット演算」（p.12-12）の説明をご覧ください。
参照	bit.bitand, bit.bitxor

例	10 進数 10（2 進数 1010）と 10 進数 9（2 進数 1001）の論理和（OR）をとると 10 進数 11（2 進数 1011）が得られます。
	`value = bit.bitor(10, 9)`
	`print(value)`
	Output: `1.100000e+01`

bit.bitxor	
機能	2 つの数値間でビット単位の排他論理和（XOR）を計算します。
使用法	`value = bit.xor(value1, value2)` `value1` XOR 演算の対象となる第 1 引数 `value2` XOR 演算の対象となる第 2 引数 `value` XOR 演算の返り値
備考	• この関数は 2 つの数値間でビット単位の排他論理和（XOR）を計算します。 • value1 と value2 の少数部を切り捨てて整数値として扱います。返り値も整数です。 • さらに詳しくは「論理演算とビット演算」（p.12-12）の説明をご覧ください。
参照	bit.bitand, bit.bitor

例	10 進数 10（2 進数 1010）と 10 進数 9（2 進数 1001）の排他論理和（XOR）をとると 10 進数 3（2 進数 0011）が得られます。
	`value = bit.bitxor(10, 9)`
	`print(value)`
	Output: `3.000000e+00`

bit.clear	
機能	index で指定された位置のビットをクリアします。
使用法	`value = bit.clear(value1, index)` `value1` 与えられた引数 `index` クリアするビットの位置を示すインデックス（1 ～ 32） `value` 操作の結果を示す返り値
備考	• index で指定された位置のビットをクリアします。 • value1 の少数部を切り捨てて整数として扱います。返り値も整数です。 • 与えられた数値の最下位ビットのインデックス が 1、最上位ビットのインデックス が 32 になります。 • さらに詳しくは「論理演算とビット演算」（p.12-12）の説明をご覧ください。
参照	bit.get, bit.getfield, bit.set, bit.setfield, bit.test, bit.toggle

例	10 進数 15 の 2 進数表現は 1111 ですから、インデックス 2 を指定してビットをクリアすると返り値の 10 進表現は 13（2 進数 1101）になります。
	`value = bit.clear(15, 2)`
	`print(value)`
	Output: `1.300000e+01`

bit.get	
機能	index で指定した位置のビットの重み付きの値を返します。
使用法	`value = bit.get(value1, index)` `value1` 与えられた引数 `index` 取り出すビットの位置を示すインデックス（1 ～ 32） `value` ビットに重率をかけた返り値
備考	• value1 の指定されたインデックス位置における値を返します。指定したビット以外をすべてゼロにセットしたときの value1 の値を返すことになります。 • 与えられた引数の少数部を切り捨てて整数化してから、index で指定された位置のビットを取り出します。与えられた数値の最下位ビットのインデックスが 1、最上位ビットのインデックスが 32 になります。 • index で指定した位置のビットが 0 にセットされていると返り値も 0 になります。 • さらに詳しくは「論理演算とビット演算」（p.12-12）の説明をご覧ください。
参照	bit.clear, bit.getfield, bit.set, bit.setfield, bit.test, bit.toggle

例	10 進数 10 の 2 進数表現は 1010 です。インデックス位置 4 のビットをこの関数で取得すると 10 進数 8 が返されます。
	`value = bit.get(10, 4)`
	`print(value)`
	Output: `8.000000e+00`

bit.getfield	
機能	指定したインデックス位置で始まるビットフィールドを返します。
使用法	`value = bit.getfield(value1, index, width)` `value1` 与えられた引数 `index` 最初のビットの位置を示すインデックス：1 ～（33 から width を引いた値） `width` フィールド幅 - フィールドに含めるビットの数（1 ～ 24） `value` ビットフィールドが返す値
備考	• ビットフィールドとは隣接したビットのグループを意味します。この関数は value1 の index 位置から始まるビットフィールドを切り出します。index は切り出すフィールドの最下位ビットの位置を示します。切り出すフィールドの幅を width で指定します。 • 与えられた引数の少数部を切り捨てて整数化してからビットフィールドの切り出しを行います。 • 与えられた引数の最下位ビットの index 値が 1、最上位ビットの index 値が 32 になります。 • さらに詳しくは「論理演算とビット演算」（p.12-12）の説明をご覧ください。
参照	bit.clear, bit.get, bit.set, bit.setfield, bit.test, bit.toggle

例	10 進数 13 の 2 進数表現は 1101 です。インデックス位置 2 から 3 ビット幅のフィールドを切り出すと 2 進数 110 が得られます。したがって返り値は 10 進数 6（2 進数 110）となります。
	`value = bit.getfield(13, 2, 3)`
	`print(value)`
	Output: `6.000000e+00`

bit.set	
機能	インデックスで指定された位置のビットをセットします。
使用法	`value = bit.set(value1, index)` `value1` 与えられた引数 `index` セットするビットの位置を示すインデックス（1 ～ 32） `value` ビットがセットされた後の新しい値
備考	• この関数は、value1 の index で指定したビットをセットされた後の新しい値を返します。index の取り得る値は 1 ～ 32 までです。指定された値（value1）の最下位ビットの index が 1、最上位ビットの index が 32 になります。 • value1 の少数部を切り捨てて整数として扱います。 • さらに詳しくは「論理演算とビット演算」（p.12-12）の説明をご覧ください。
参照	bit.clear, bit.get, bit.getfield, bit.setfield, bit.test, bit.toggle

例	10 進数 8 の 2 進数表現は 1000 です。index に 3 を指定した場合の返り値は 10 進数 12（2 進数 1100）となります。
	`value = bit.set(8, 3)`
	`print(value)`
	Output: `1.200000e+01`

bit.setfield	
機能	インデックスで指定された位置のビットフィールドを上書きします。
使用法	`value = bit.setfield(value1, index, width, fieldvalue)` `value1` 与えられた引数 `index` フィールドの最下位ビットの位置を示すインデックス：1 ～（33 から width を引いた値） `width` フィールド幅 - フィールドに含めるビット数（1 ～ 24） `fieldvalue` フィールドを上書きする値 `value` フィールド上書き後の新しい返り値
備考	• この関数は、value1 の index から始まるビットフィールドを上書きした値（value）を返します。index は上書きするフィールドの最下位ビットの位置を示し、ここから始まる幅 width のビットフィールドに fieldvalue で指定する値を上書きします。value1 の最下位ビット位置が index の 1 となり、最上位ビットの index 値は 32 となります。 • value1 と fieldvalue の小数部分を切り捨てて整数化してから、ビットフィールドを上書きします。 • fieldvalue の幅が width よりも広かった場合は、fieldvalue の余分なビットを最上位ビット側から切り捨てます。たとえば、width に 4 ビットを指定したときに fieldvalue の 2 進表現が 11110（5 ビット）であったとすれば、最上位ビットを切り捨てた 2 進数 1110 を fieldvalue の値として使用します。 • さらに詳しくは「論理演算とビット演算」（p.12-12）の説明をご覧ください。
参照	bit.clear, bit.get, bit.getfield, bit.set, bit.test, bit.toggle

例	10 進数 15 の 2 進数表現は 1111 です。インデックス位置 2 から始まるフィールドを 10 進数 5（2 進数 101）で上書きすると、返り値（value）は 10 進数 11（2 進数 1011）となります。
	`value = bit.setfield(15, 2, 3, 5)`
	`print(value)`
	Output: `1.100000e+01`

bit.test	
機能	インデックスで指定した位置のビットの値をブール値（true/false）として返します。
使用法	`value = bit.test(value1, index)` `value1` 与えられた引数 `index` テストの対象となるビットの位置を示すインデックス（1 ～ 32） `value` 指定したビットを 10 進数で表現した返り値
備考	• テストの対象となったビットの値を返します。与えられた数値の最下位ビットのインデックス が 1、最上位ビットのインデックスが 32 となります。 • value1 の少数部を切り捨てて整数として扱います。value1 の index で指定されたビットが 0 にセットされていた場合は false を返し、1 にセットされていた場合は true を返します。 • index の値が value1 のビット数を超えていた場合は false を返します。 • さらに詳しくは「論理演算とビット演算」（p.12-12）の説明をご覧ください。
参照	bit.clear, bit.get, bit.getfield, bit.set, bit.setfield, bit.toggle

例	10 進数 10 の 2 進数表現は 1010 です。インデックス位置 4 のビットをこの関数でテストすると Boolean 値 true を返します。
	`value = bit.test(10, 4)`
	`print(value)`
	Output: `true`

bit.toggle	
機能	インデックスで指定した位置のビットを反転させます。
使用法	`value = bit.toggle(value1, index)` `value1` 与えられた引数 `index` 反転させるビットの位置を示すインデックス（1 ～ 32） `value` ビット反転後の新しい返り値
備考	• value1 の指定ビットを反転させた後の値を返します。 • value1 の少数部を切り捨てて整数として扱います。返される 10 進数も整数です。与えられた数値の最下位ビットのインデックス が 1、最上位ビットのインデックスが 32 となります。 • value1 の index で指定された位置のビットが反転（0 → 1/1 → 0）します。 • さらに詳しくは「論理演算とビット演算」（p.12-12）の説明をご覧ください。
参照	bit.clear, bit.get, bit.getfield, bit.set, bit.setfield, bit.test

例	10 進数 10 の 2 進数表現は 1010 です。インデックス位置 3 のビットをこの関数で反転させると 10 進数 14（2 進数 1110）を返します。
	`value = bit.toggle(10, 3)`
	`print(value)`
	Output: `1.400000e+01`

delay 関数

この関数を使用してシステムの動作を指定した時間だけ休止させます。デバイスを一定時間特定の電圧や電流に曝して安定化させるのが典型的な利用法です。

delay	
機能	システム動作を一定時間ディレイさせます。
使用法	`delay(seconds)` `seconds` ディレイを秒で設定します（最長 100000 秒）。
備考	• この関数は指定した秒数だけのディレイを発生します。0 秒のディレイを発生させることはできません。 • 50 μs 未満のディレイを指定した場合、システムのオーバーヘッドが支配因子となって実際には 50 μs（典型値）程度のディレイが発生します。50 μs 以上のディレイを指定すると、実際には指定した値に 10 μs（典型値）程度を加算したディレイが発生します。

例	SMU A 出力を 1V に設定し、DUT をこの電圧で 50ms 間だけ馴らしてから出力を OFF にします。
	`smua.source.levelv = 1.0`
	`delay(0.050)`
	`smua.source.off()`

digio 関数と属性値

このグループに属する関数と属性値を使用してデジタル I/O ポートの読み込み / 書き込み、およびトリガ操作を行います。

注意 **デジタル I/O ラインは入力と出力両方に使用できます。入力として使用するすべてのデジタル I/O ラインには “1” を書き込んでおかなければなりません。この状態でラインを外部から駆動して Low に変化させると、そのラインに該当するコマンドが “0” を読み取ります。**

digio.readbit	
機能	1 本のデジタル I/O ラインから値を読み込みます。
使用法	`data = digio.readbit(line)` `line` 読み込みの対象となるデジタル I/O ライン番号（1 ～ 14）
備考	返り値の “0” はラインが Low になっていることを意味し、“1” はラインが High であることを示します。
詳細	セクション 10「デジタル I/O ポート」をご覧ください。
参照	digio.readport, digio.writebit, digio.writeport

例	ライン 4 を読み込みます（このラインは予め High にセットされているものとします）。
	`data = digio.readbit(4)`
	`print(data)`
	Output: `1.000000e+00`

digio.readport	
機能	デジタル I/O ポート読み込みます。
使用法	`data = digio.readport()`
備考	返り値を 2 進数で表現したときのビットが I/O ポートの入力パターンを表します。2 進表現の最下位ビットがライン 1 に対応し、ビット 14 がライン 14 に対応します。たとえば、返り値が 170 であったとすれば、その 2 進表現は 00000010101010 となりますから、ライン 2、4、6、8 が High(1)、その以外の 10 本のラインが Low（0）であることを表します。
詳細	セクション 10「デジタル I/O ポート」をご覧ください。
参照	digio.readbit, digio.writebit, digio.writeport

例	I/O ポートを読み込みます（ライン 2、4、6、8 が予め High にセットされているものとします）。
	`data = digio.readport()`
	`print(data)`
	Output: `1.700000e+02` (binary 10101010)

digio.trigger[N].assert	N はデジタル I/O トリガラインの番号（1 ～ 14）
機能	デジタル I/O ラインへのトリガ出力をアサートします。
使用法	`digio.trigger[N].assert()`
備考	トリガをアサートする時間は設定されているパルス幅によって決まります。
詳細	セクション 10「デジタル I/O ラインのコントロール」をご覧ください。
参照	digio.trigger[N].pulsewidth
例	I/O ライン 2 のトリガをアサートします。 `digio.trigger[2].assert()`

digio.trigger[N].clear	N はデジタル I/O トリガラインの番号（1 ～ 14）
機能	デジタル I/O ラインのトリガイベントをクリアします。
使用法	`digio.trigger[N].clear()`
備考	トリガイベント検出回路は、前回の digio.trigger[line].wait 呼び出し以後にトリガイベントが検出されたか否かを記憶しています。この関数はトリガイベント検出回路をクリアしてトリガラインのそれまでの履歴を破棄します。
詳細	セクション 10「デジタル I/O ラインのコントロール」をご覧ください。
参照	digio.trigger[N].wait
例	I/O ライン 2 のトリガイベントをクリアします。 `digio.trigger[2].clear()`

digio.trigger[N].mode	N はデジタル I/O トリガラインの番号（1 ～ 14）
属性値	トリガの操作と検出モード。
使用法	`tmode = digio.trigger[N].mode` -- トリガモード読み込み `digio.trigger[N].mode = tmode` -- トリガモード書き込み tmode に次のいずれかの値をセットします。 1 or `digio.TRIG_FALLING` 入力 : 下降エッジトリガを検出します。 出力 : TTL-Low のトリガパルスをアサートします。 2 or `digio.TRIG_RISING` 入力 : 上昇エッジトリガを検出します。 出力 : TTL-High のトリガパルスをアサートします。 3 or `digio.TRIG_EITHER` 入力 : 上昇または下降エッジトリガを検出します。 出力 : TTL-Low のトリガパルスをアサートします。 5 or `digio.TRIG_SYNCHRONOUS` 入力 : 下降エッジトリガを検出してラッチし、ラインを Low に駆動します。 出力 : TTL-Low のトリガパルスをアサートします。

備考	• tmode を指定するには数字（1、2、3、5）または定義済み定数（「使用法」参照）を使用します。 • トリガモードの読み取りではモードが数値として返されます。
詳細	セクション 10「デジタル I/O ラインのコントロール」参照。
例	I/O ライン 4 のトリガモードを TRIG_RISING へ設定します。 `digio.trigger[4].mode = 2`

digio.trigger[N].pulsewidth	N はデジタル I/O トリガラインの番号（1 ～ 14）
属性値	トリガラインがトリガ出力をアサートしている時間の長さ。
使用法	`width = digio.trigger[N].pulsewidth`-- パルス幅を読み取る `digio.trigger[N].pulsewidth = width`-- パルス幅を書き込む `width`　　パルス幅（秒）
備考	• トリガラインが少なくとも一定時間（多分それよりも若干長い時間）アサートされることが保証されます。 • パルス幅を 0（秒）に設定するとトリガが時間無制限にアサートされます。 • パルス幅のデフォルト時間は 10 μs です。
詳細	セクション 10「デジタル I/O ラインのコントロール」参照。
参照	digio.trigger[N].release
例	トリガライン 4 のパルス幅を 20 μs に設定します。 `digio.trigger[4].pulsewidth = 20e-6`

digio.trigger[N].release	N はデジタル I/O トリガラインの番号（1 ～ 14）
機能	無限長アサートされたトリガやラッチされたトリガを解放します。
使用法	`digio.trigger[N].release()`
備考	パルス幅を無限長としてアサートされていたトリガを解放します。また、同期モードトリガの受信に応答してアサートされていたトリガを解放します。このコマンドの影響を受けるのは指定されたトリガライン（N）だけです。
詳細	セクション 10「デジタル I/O ラインのコントロール」参照。
参照	digio.trigger[N].pulsewidth
例	トリガライン 4 を解放します。 `digio.trigger[4].release()`

digio.trigger[N].wait	N はデジタル I/O トリガラインの番号（1 ～ 8）
機能	トリガの発生を待ちます。
使用法	`triggered = digio.trigger[N].wait(timeout)` `timeout` タイムアウト時間（秒）を設定します。 `triggered` トリガを検出すると ‘true’ を返し、タイムアウト時間終了までにトリガを検出しなかった場合は ‘false’ を返します。
備考	この関数はタイムアウト時間が終了するまで入力トリガを待ち続けます。最後に digio.trigger[N].wait または digio.trigger[N].clear を呼び出してから以後 1 つでもトリガを検出すると関数は直ちにリターンします。この関数を使用してトリガ待ちを行った後はイベントディテクタが自動的にリセットされてリアームされます。この動作は検出したイベントの数によらず同様です。
詳細	セクション 10「デジタル I/O ラインのコントロール」をご覧ください。
参照	digio.trigger[N].clear

例	トリガライン 4 上のトリガ発生を最長 3 秒間待ち続け、トリガを検出したならばディスプレイに表示します。
	`triggered = digio.trigger[4].wait(3)`
	`print(triggered)`
	Output: `false` (トリガ検出なし)
	`true` (トリガ検出)

digio.writebit	
機能	デジタル I/O ラインを High または Low にセットします。
使用法	`digio.writebit(bit, data)` `bit` デジタル I/O ラインの番号（1 ～ 14） `data` 該当ビットに書き込む値：0（Low）または 1（High）
備考	• 出力ライン digio.writeprotect 属性値による書き込み保護が設定されている場合、このコマンドは無視されます。 • reset 関数はデジタル I/O ラインの現在の状態に影響を与えません。
詳細	セクション 10「デジタル I/O ポート」をご覧ください。
参照	digio.readbit, digio.readport, digio.writeport
例	デジタル I/O ライン 4 を Low（0）にセットします。 `digio.writebit(4, 0)`

digio.writeport	
機能	全部のデジタル I/O ラインへ書き込みを行います。
使用法	digio.writeport(data) data　　ポートへ書き込む値：0 ～ 16383
備考	• データの 2 進表現が I/O ポートに書き込まれる出力パターンを表します。たとえば、データ値が 170 であったとすれば、その 2 進表現は 00000010101010 となりますから、ライン 2、4、6、8 が High(1)、その以外の 10 本のラインが Low（0）にセットされます。 • 書き込み保護されているラインはこのコマンドによって変更されません（digio.writeprotect 参照）。 • reset 関数はデジタル I/O ラインの現在の状態に影響を与えません。
詳細	セクション 10「デジタル I/O ポート」をご覧ください。
参照	digio.readbit, digio.readport, digio.writebit
例	デジタル I/O ラインの 1 から 8 までを High にセットします（2 進表現 00000011111111）。 digio.writeport(255)

digio.writeprotect	
属性値	書き込み保護マスクを設定して、ラインの状態が digio.writebit と digio.writeport によって変更されるのを防止します。
使用法	mask = digio.writeprotect　　-- 書き込み保護マスクの状態を読み出します。 digio.writeprotect = mask　　-- 書き込み保護マスクを書き込みます。 mask　　この値によって書き込み保護のビットパターンを指定します。
備考	• 1 にセットしたビットに対応するラインに書き込み保護が設定されます。 • mask の値を 2 進表現したパターンが I/O ポートに設定されるマスクを表します。たとえば、mask の値が 7 であるとすれば、その 2 進表現は 00000000000111 となりますから、ライン 1、2、3 に書き込み保護が設定されます。
詳細	セクション 10「デジタル I/O ポート」をご覧ください。
参照	digio.writebit, digio.writeport
例	ライン 1、2、3、4 に書き込み保護を設定します。 digio.writeprotect = 15

display 関数と属性値

このグループに属する関数と属性値を使用してディスプレイの各種操作（詳しくは第 14 章参照）を行います。

display.clear

機能	ディスプレイの全部の行をクリアします。
使用法	`display.clear()`
備考	• ディスプレイをユーザ画面に切換えて全画面をクリアします。表示ランプの状態には影響しまん。 • display.clear()、display.setcursor、および display.settext 関数はオーバーラップ型の、他の動作をブロックしないタイプのコマンドです。つまり、スクリプトはこれらのコマンドの動作完了を待たずに次の動作を実行します。このような非ブロックタイプの関数は性能上の考慮から、ディスプレイを即時には更新せず、まず裏で処理を完了させておいてから処理時間に余裕が生ずるのを待って画面を更新します。
詳細	「ディスプレイのクリア」（p.14-5）の説明をご覧ください。
参照	display.setcursor, display.settext

display.getannunciators

機能	現在点灯している表示ランプの種類を読み取ります。
使用法	`annun = display.getannunciators()` `annun` 現在点灯している表示ランプのビットマップ値を返します。
備考	この関数は現在どの表示ランプが点灯しているのかを示すビットマップ値を返します。返り値を 16 ビットの 2 進数表現に変換したものがビットマップです。たとえば、返り値が 1028 であったとすれば、これに対応する 2 進数は次のように表現されます。 0000010000000100 このビットマップではビット 3 と 11 がセットされています。これを次の表と比較することにより、ビット 3 と 11 が点灯中のどの表示ランプに対応するかが分ります（4W と REM）。ビット 3 と 11 に 2 進加重して加算した値が返り値（1028）になっていることに注意してください。

表示ランプ	ビット	対応する 10 進数値	表示ランプ	ビット	対応する 10 進数値
FILT	1	1	EDIT	9	256
MATH	2	2	ERR	10	512
4W	3	4	REM	11	1024
AUTO	4	8	TALK	12	2048
ARM	5	16	LSTN	13	4096
TRIG	6	32	SRQ	14	8192
* (star)	7	64	REAR	15	16384
SMPL	8	128	REL	16	32768

詳細	「表示ランプ」（p.14-11）の説明をご覧ください。

例	現在点灯している表示ランプの種類を読み取ります。
	`annun = display.getannunciators()`
	`print(annun)`
	Output: `1.280000e+03`
	返り値 1280 に対応する 2 進数表現は 0000010100000000 となり、ビット 9 と 11 がセットされています。これを「備考」で説明した表と比較することにより、REM と EDIT が点灯していることが分ります。

display.getcursor	
機能	ユーザディスプレイ上の現在のカーソル位置を読み取ります。
使用法	`row, column, style = display.getcursor()` `row` 現在のカーソル位置の行番号を返します。 `column` 現在のカーソル位置のカラム番号を返します。 `style` カーソルのスタイルを返します。
備考	• この関数はディスプレイをユーザ画面に切換えると共に、カーソル現在位置の行、カラム番号、およびスタイルを返します。 • 行番号の返り値は 1（上の行）または 2（下の行）のいずれかです。 • カーソルが上の行にあるときのカラム番号返り値は 1 ～ 20 までの値をとり、下の行にあるときのカラム番号返り値は 1 ～ 32 までの値をとります。カラム番号はディスプレイ上の左から右に向けて番号が増加します。 • カーソルスタイル返り値は 0（非表示）または 1（点滅）のいずれかです。
詳細	「カーソル位置」（p.14-5）の説明をご覧ください。
参照	display.gettext, display.screen, display.setcursor, display.settext

例	カーソル位置（行、カラム）を読み取ります。
	`row, column = display.getcursor()`
	`print(row, column)`
	Output: `1.000000e+00 3.000000e+00`
	上の出力はカーソル位置が Row 1 の Column 3 であることを示します。

display.getlastkey

機能	いちばん最近押されたキーのキーコードを返します。
使用法	`key = display.getlastkey()`

備考	• この読み取り専用関数はいちばん最近押されたキーのキーコードを返します。key には次のいずれかの値が返されます。
	`0 (display.KEY_NONE)` `82 (display.KEY_ENTER)`
	`65 (display.KEY_RANGEUP)` `83 (display.KEY_MEASB)`
	`67 (display.KEY_RELB)` `84 (display.KEY_DIGITSB)`
	`68 (display.KEY_MENU)` `85 (display.KEY_RECALL)`
	`69 (display.KEY_MODEA)` `86 (display.KEY_MEASA)`
	`70 (display.KEY_RELA)` `87 (display.KEY_DIGITSA)`
	`71 (display.KEY_RUN)` `90 (display.KEY_LIMITB)`
	`72 (display.KEY_DISPLAY)` `91 (display.KEY_SPEEDB)`
	`73 (display.KEY_AUTO)` `92 (display.KEY_TRIG)`
	`74 (display.KEY_FILTERB)` `93 (display.KEY_LIMITA)`
	`75 (display.KEY_EXIT)` `94 (display.KEY_SPEEDA)`
	`76 (display.KEY_SRCB)` `95 (display.KEY_LOAD)`
	`77 (display.KEY_FILTERA)` `97 (display.WHEEL_ENTER)`
	`78 (display.KEY_STORE)` `103 (display.KEY_RIGHT)`
	`79 (display.KEY_SRCA)` `104 (display.KEY_LEFT)`
	`80 (display.KEY_CONFIG)` `114 (display.WHEEL_RIGHT)`
	`81 (display.KEY_RANGEDOWN)`
	• いちばん最近押された前面パネルキーのキーコード履歴は 260x 型によって保持され、装置への電源投入時（またはローカルからリモートへの移行時）にキーコードは 0（display.KEY_NONE）にセットされます。
	• SMU A と SMU B の OUTPUT ON/OFF キー操作をこの関数で追跡することはできません。
	• 通常、EXIT/LOCAL キーが押されるとスクリプトの実行は中断されます。EXIT キーと共にこの関数を用いるためには、display.locallockout を使用する必要があります。

詳細	「キーコードを送る」（p.14-16）をご覧ください。
参照	display.sendkey, display.locallockout

例	前面パネル上の MENU キーを押してから次のコードを送ります。
	`key = display.getlastkey()`
	`print(key)`
	Output: `6.800000e+01`

display.gettext

機能

現在ディスプレイに表示されているテキストを読み込みます。

使用法

この関数は次の 5 通りの方法で使用します。

```
text = display.gettext()
text = display.gettext(embellished)
text = display.gettext(embellished, row)
text = display.gettext(embellished, row, column_start)
text = display.gettext(embellished, row, column_start, column_end)
```

`embellished`	単純文字列として返すときは false、すべての文字コードを含めて返すときは true に設定します。
`row`	1 または 2 を指定してどちらの行を読み出すかを選択します。このパラメータを使用しなかった場合は両方の行を読み出します。
`column_start`	テキストを読み出す最初の桁を指定します（デフォルトは 1）。
`column_end`	テキストを読みだす最後の桁を指定します（デフォルト値：Row 1 は 20、Row 2 は 32）。

注： 使用できる桁数はどちらの行を指定するかによって異なります。Row 1 の有効桁数は 1 ～ 20、Row 2 の有効桁数は 1 ～ 32 です。

備考

- パラメータを何も与えずにコマンドを呼び出すと、ディスプレイ上の両方の行を返します。その際、上の行の終わりと下の行の始まりを区別するために文字コード $N が返り値に含まれます。
- embellished を true にセットしておくと、メッセージに加えて他のすべての文字コードも返されます。embellished を false にセットしておくと、メッセージと $N 文字コードのみが返されます。文字コードの詳細については display.settext 関数の項をご覧ください。
- この関数を実行してもディスプレイはユーザ画面に切り換わりません。テキストはその時アクティブな画面から読み取られます。

詳細

「テキストメッセージの表示」（p.14-6）の説明をご覧ください。

参照

display.getcursor, display.setcursor, display.settext

例

ディスプレイの両方の行からすべてのテキストを読み取ります。

```
text = display.gettext()
print(text)
```

Output: `User Screen $N`

上の出力例は、上の行に “User Screen” メッセージが表示され、下の行は空白であることを示します。

display.inputvalue

機能

オペレータが編集できる書式化された入力フィールドを表示します。

使用法

この関数は次の 4 通りの方法で使用します。

```
value = display.inputvalue(format)
value = display.inputvalue(format, default)
value = display.inputvalue(format, default, min)
value = display.inputvalue(format, default, min, max)
```

format	以下の記号を使用して入力フィールドで使用する書式を定義します。“0”、少数点（.）、符号（+）、指数記号（E）。
default	パラメータにデフォルト値をセットします。
min	入力を許す最小値をセットします。
max	入力を許す最大値をセットします。

備考

- この関数はテキストを使用して、ユーザ画面の現在のカーソル位置に編集可能な入力フィールドを作成します。電源投入後の最初の書き込みによってユーザ画面がクリアされます。
- 入力フィールド例：
  +0.00　　00　　+00.0000E+00　　0.00000E+0

  値フィールド：

+	正または負の入力を許すときに “+” 記号を含めます。“+” 記号を含めなかった場合は負の値を入力できなくなります。
0‘s’	値の桁位置を定義します。最高 6 個までの ‘0’ を並べることができます（上の 3 番目と 4 番目の例参照）。
.	この記号を使用すると、値が必要とする場合に少数点（.）が含まれます。

  指数フィールド（省略可）：

E	指数値を入力するときに “E” を含めます。
+	正または負の入力を許すときに “+” 記号を含めます。“+” 記号を含めなかった場合は負の指数値を入力できなくなります。
0‘s’	指数値の桁位置を定義します。

- 入力フィールドの書式を指定することに加えて、入力可能な最大値と最小値を指定できるオプションを使用することができます。ただし、値フィールドに “+” 記号を使用しなかった場合は最小値リミットとして負の値を設定することはできません。“+” 記号を使用した場合は負の最小値（たとえば -2）を指定することができます。
- デフォルト値を指定するオプションも用意されています。このコマンドを実行したときに最初にそのフィールドに表示される値がデフォルト値となります。
- この関数呼び出しの前に、オペレータに入力を要求するプロンプトメッセージを表示するようにしてください。編集フィールドの位置に合わせてカーソルを移動させてください。
- 入力可能な値は ± 1e37 の範囲に制限されています。

備考（続き）	• このコマンドの送信後、スクリプトは停止状態になってオペレータが値を入力して ENTER を押すのを待ちます。 • リミットが設定されている場合、オペレータは最大値、最小値リミットの範囲を超える値を入力できません。 • 正負両方の入力を許すように設定されている場合（値フィールド / 指数フィールドに “+” 記号を使用）、ゼロ以外の値 / 指数の極性記号上にカーソルを移動させてホイールを回転するだけで符号を反転させることができます。ホイールを回転させて値 / 指数を増減させてゆくと、ゼロを超えるタイミングで自動的に極性符号も反転します。値または指数がゼロ（つまり +00）のときは常に正として取り扱われ、符号を反転することはできません。 • このコマンドを送信後に EXIT キーを押すと value には nil が返されます。
詳細	「パラメータ入力プロンプト」（p.14-10）参照。
参照	display.prompt, display.setcursor, display.settext

例	オペレータに入力させる編集フィールド（“+0.50”）を表示します。
	入力範囲は -0.10 ～ +2.00、デフォルト値 は 0.50。
	`display.clear()`
	`value = display.inputvalue("+0.00", 0.5, -0.1, 2.0)`

display.loadmenu.add

機能	“LOAD TEST” メニューのサブメニューである “USER TESTS” にメニュー項目を追加します。

使用法	この関数は次の 2 通りの方法で使用します。
	`display.loadmenu.add(displayname, chunk)`
	`display.loadmenu.add(displayname, chunk, memory)`
	`displayname` メニュー項目として表示する名称
	`chunk` 実行するコード（チャンク）を指定
	`memory` chunk と displayname を不揮発性メモリに保存するか否かを指定
	memory に次のいずれかの値をセットします。
	0 または `display.DONT_SAVE`
	1 または `display.SAVE`
	memory のデフォルト設定は display.SAVE です。

備考	• この関数は “LOAD TEST” メニューのサブメニューである “USER TESTS” にメニュー項目を追加します。追加した項目を前面パネルから選択し、RUN キーを押すことにより対応するチャンクが実行されます。 • チャンクはスクリプト、関数、変数、およびコマンドによって構成されています。memory に display.SAVE をセットしておくと、コマンドがチャンクと共に不揮発性メモリに保存されます。ただし、このチャンクが使用するスクリプト、関数、および変数は display.SAVE によって保存されません。関数と変数はスクリプトと共に保存する必要があります（「ユーザスクリプトの保存」、p.14-13、参照）。不揮発性メモリに保存されなかったスクリプトは 260x 型の電源が切れると同時に失われます。下の 例 1 を参照。 • “USER TESTS” を選択するとメニュー項目はアルファベット順に表示され、メニューを追加した順序には影響を受けません。

詳細	「テストメニューのローディング」（p.14-13）参照。
参照	display.loadmenu.delete

例	**例 1**– “DUT1” という名前の関数を含むスクリプトがすでに 260x 型へローディングされており、スクリプトは不揮発性メモリに保存されていないものとします。ここで、“Test” という名前のテストを USER TESTS メニューに追加したいとします。このテストは関数 “DUT1” を実行してブザーを鳴動させます。
	以下のコマンドはメニューに項目 “Test” を追加し、チャンクを定義し、続いて displayname と chunk を不揮発性メモリに保存します。
	`display.loadmenu.add("Test", "DUT1() beeper.beep(2, 500)", display.SAVE)`
	前面パネルの USER TESTS メニューから “Test” を実行すると、“DUT1” という名前の関数が実行されてブザーを 2 秒間鳴らします。
	ここで、一旦 260x 型の電源を切ってから再投入するものとします。スクリプトは不揮発性メモリに保存されていませんでしたから、関数 “DUT1” はこの操作で失われます。再び前面パネルから “Test” を実行すると、ブザーは鳴りますが、“DUT1” はすでにチャンクに存在しませんから実行されません。
	例 2 – 前面パネルの “USER TESTS” ロードメニューに “Part1” という項目（チャンク “testpart([[Part1]], 5.0)” に対応）を追加して揮発性メモリに保存します。
	`display.loadmenu.add("Part1", "testpart([[Part1]], 5.0)", display.SAVE)`

display.loadmenu.delete	
機能	“LOAD TEST” メニューのサブメニューである “USER TESTS” からメニュー項目を削除します。
使用法	`display.loadmenu.delete(displayname)` `displayname` メニューから削除する名前
備考	この関数を使用して、前面パネルの LOAD TEST メニューのサブメニューである USER TESTS から項目（displayname）を削除します。
詳細	「テストメニューのローディング」（p.14-13）参照。
参照	display.loadmenu.add
例	前面パネルの “USER TESTS” ロードメニューから項目 “Part1” を削除します。 `display.loadmenu.delete("Part1")`

display.locallockout	
属性値	LOCAL キー使用不可。
使用法	`lockout = display.locallockout` -- ロックアウトの状態読み出し `display.locallockout = lockout` -- ロックアウトの状態書き込み lockout に次のいずれかの値をセットします。 0 or `display.UNLOCK`　LOCAL キーロックアウト解除 1 or `display.LOCK`　LOCAL キーをロックアウト
備考	display.locallockout に display.LOCK をセットしておくと、ユーザが LOCAL キーを押してもリモート操作を中断できません。この属性値に display.UNLOCK をセットしておけば、LOCAL キーを押すことによりスクリプト / リモート操作を中断させることができます。
詳細	「LOCAL ロックアウト」（p.14-12）参照。
例	前面パネルの LOCAL キーが機能しないように設定します。 `display.locallockout = display.LOCK`

display.menu	
機能	前面パネルディスプレイにメニューを表示します。
使用法	`selection = display.menu(name, items)` `name`　上の行に表示するメニュー名 `items`　下の行に表示するメニュー名
備考	• メニューでは名前を表す name 文字列が上の行に表示され、選択可能な items リストが下の行に表示されます。メニュー項目（items）はそれぞれの項目を表す 1 単語を空白文字で区切ります。上の行に表示する name は 20 文字以内でなければなりません。 • このコマンドの送信後、スクリプトは動作を一時停止してオペレータがメニュー項目を選択するのを待ちます。オペレータはホイールを回転させるか、またはカーソルキーを使用して点滅カーソルを希望の項目に移動させ、ホイールまたは ENTER キーを押して確定させます。選択が確定すると、関数はその項目を表す文字列を返します。 • メニューが表示されている間に EXIT キーを押してもスクリプトは中断されませんが、関数は nil を返します。この操作に合わせてスクリプトを中断したい場合は nil が返されたタイミングで exit 関数を呼び出してください。
詳細	「メニュー」（p.14-8）参照。

例	3 項目から構成されるメニューを表示します。2 番目のメニュー項目が選択されると、selection に “Test2” が返されます。
	`selection = display.menu("Sample Menu", "Test1 Test2 Test3")`
	`print(selection)`
	Output: `Test2`

display.prompt

機能

前面パネルにプロンプトを表示してユーザにパラメータ入力を要求します。

使用法

この関数は次の 4 通りの方法で使用します。

```
value = display.prompt(format, units, help)
value = display.prompt(format, units, help, default)
value = display.prompt(format, units, help, default, min)
value = display.prompt(format, units, help, default, min, max)
```

format	以下の記号を使用して入力フィールドで使用する書式を定義します。“0”、少数点（.）、極性符号（+）、指数記号（E）。
units	上の行に表示する単位テキスト文字列を設定します（8 文字以内）。
help	下の行に表示するテキスト文字列（32 文字以内）。
default	パラメータにデフォルト値をセットします。
min	入力を許す最小値をセットします。
max	入力を許す最大値をセットします。

備考

- この関数は現在のカーソル位置に編集可能な入力フィールドを作成し、下の行には入力プロンプトメッセージを表示します。次に示すのは入力フィールドとプロンプトの表示例です。

  0.00V
  Input 0 to +2V

- format は編集可能な入力フィールドの書式を規定します。4 種類の書式例を次に示します。

  +0.00　　00　　+00.0000E+00　　0.00000E+0

  値フィールド：

+	正または負の入力を許すときに “+” 記号を含めます。“+” 記号を含めなかった場合は負の値を入力できなくなります。
0‘s’	値の桁位置を定義します。最高 6 個までの ‘0’ を並べることができます（上の 3 番目と 4 番目の例参照）。
.	この記号を使用すると、値が必要とする場合に少数点（.）が含まれます。

  指数フィールド（省略可）

E	指数値を入力するときに “E” を含めます。
+	正または負の入力を許すときに “+” 記号を含めます。“+” 記号を含めなかった場合は負の指数値を入力できなくなります。
0‘s’	指数値の桁位置を定義します。

備考（続き）	• units は値が使用する単位を示す文字列（“V” または “A”）、help は下の行に表示されるプロンプトメッセージです。 • 入力フィールドの書式を指定することに加えて、入力可能な最大値と最小値を指定できるオプションを使用できます。ただし、値フィールドに “+” 記号を使用しなかった場合は最小値リミットとして負の値を設定することはできません。“+” 記号を使用した場合は負の最小値（たとえば -2）を指定することができます。 • デフォルト値を指定するオプションも用意されています。このコマンドを実行したときに最初にそのフィールドに表示される値がデフォルト値となります。 • この関数呼び出しの前に、オペレータに入力を要求するプロンプトメッセージを表示するようにしてください。編集フィールドの位置に合わせてカーソルを移動させてください。 • 入力可能な値は ± 1e37 の範囲に制限されます。 • このコマンドの送信後、スクリプトは停止状態になってオペレータが値を入力して ENTER を押すのを待ちます。 • リミットが設定されている場合、オペレータは最大値と最小値リミットの範囲を超える値を入力できません。 • 正負両方の入力を許すように設定されている場合（値フィールド / 指数フィールドに “+” 記号を使用）、ゼロ以外の値 / 指数の極性記号上にカーソルを移動させてホイールを回転するだけで符号を反転させることができます。ホイールを回転させて値 / 指数を増減させてゆくと、ゼロを超えるタイミングで自動的に極性符号も反転します。値または指数がゼロ（つまり +00）のときは常に正として取り扱われ、符号を反転することはできません。 • このコマンドを送信後に EXIT キーを押すと value には nil が返されます。
詳細	「パラメータ入力プロンプト」（p.14-10）参照。
参照	display.inputvalue
例	プロンプトを表示してオペレータに電圧値の入力を要求します。 有効な入力範囲は 0 ～ +2.00、デフォルト値は 0.50。 `value = display.prompt("0.00", "V", "Input 0 to +2V" 0.5, 0, 2)` 上記コマンドを実行すると、次の入力プロンプトが表示されます。 0.50V Input 0 to +2V

display.screen

属性値	選択されているディスプレイ画面
使用法	`displayid = display.screen` -- ディスプレイ画面の読み出し `display.screen = displayid` -- ディスプレイ画面の書き込み displayid を次のいずれかの値にセットします。 0 または display.SMUA　SMU A のソースメジャーとコンプライアンスを表示します。 1 または display.SMUB　SMU B のソースメジャーとコンプライアンスを表示します。 2 または display.SMUA_SMUB　SMU A と SMU B のソースメジャーを表示します。 3 または display.USER　ユーザ画面を表示します。

備考	この属性値を設定することによって前面パネルに表示する画面を選択します。この属性値を読み取ることにより、使用可能な 4 種類の表示画面の中でユーザがどれを最後に選択したかを知ることができます。画面の選択には数値または列挙子を指定します。
詳細	「ディスプレイ画面」（p.14-3）参照。
例	SMU A のソースメジャーとコンプライアンスリミット表示を選択します。 `display.screen = display.SMUA`

display.sendkey

機能	前面パネルの操作を模擬するキーコードを送ります。

使用法	display.sendkey(keycode)
	keycode に次のいずれかの値をセットします。
	`display.KEY_AUTO or 73` `display.KEY_OUTPUTA or 88`
	`display.KEY_CONFIG or 80` `display.KEY_OUTPUTB or 96`
	`display.KEY_DIGITSA or 87` `display.KEY_RANGEDOWN or 81`
	`display.KEY_DIGITSB or 84` `display.KEY_RANGEUP or 65`
	`display.KEY_DISPLAY or 72` `display.KEY_RECALL or 85`
	`display.KEY_ENTER or 82` `display.KEY_RELA or 70`
	`display.KEY_EXIT or 75` `display.KEY_RELB or 67`
	`display.KEY_FILTERA or 77` `display.KEY_RIGHT or 103`
	`display.KEY_FILTERB or 74` `display.KEY_RUN or 71`
	`display.KEY_LEFT or 104` `display.KEY_SPEEDA or 94`
	`display.KEY_LIMITA or 93` `display.KEY_SPEEDB or 91`
	`display.KEY_LIMITB or 90` `display.KEY_SRCA or 79`
	`display.KEY_LOAD or 95` `display.KEY_SRCB or 76`
	`display.KEY_MEASA or 86` `display.KEY_STORE or 78`
	`display.KEY_MEASB or 83` `display.KEY_TRIG or 92`
	`display.KEY_MENU or 68` `display.WHEEL_ENTER or 97`
	`display.KEY_MODEA or 69` `display.WHEEL_LEFT or 107`
	`display.KEY_MODEB or 66` `display.WHEEL_RIGHT or 114`

備考	このコマンドを送ることにより、前面パネルキーやホイールを押すことや、ホイールの左右への 1 クリックを模擬することができます。
詳細	「キーコードを送る」（p.14-16）参照。
例	RUN キーの「Press」を模擬します。 `display.sendkey(display.KEY_RUN)`

display.setcursor	
機能	カーソル位置を設定します。

使用法	この関数は次の 2 通りの方法で使用します。
	`display.setcursor(row, column)`
	`display.setcursor(row, column, style)`
	`row` カーソル位置の行番号（1 または 2）を指定します。
	`column` カーソル位置の桁番号を指定します。row が 1 の場合は 1 ～ 20、row が 2 の場合は 1 ～ 32 までの値を指定できます。
	`style` カーソルのスタイルを非表示（0）または点滅（1）のいずれかに指定します。

備考	• このコマンドはユーザ画面を選択し、続いてカーソルを指定の位置へ移動させます。 • row に範囲外の値を指定すると 2 行目が選択され、桁位置に範囲外の値を指定すると 20（row 1）または 32（row 2）が選択されます。 • style に範囲外の値を指定すると 0（非表示）が選択されます。 • 点滅カーソルは表示されているテキスト上に位置したときにだけ表示されます。空白文字上のカーソルは非表示状態になります。 • display.clear、display.setcursor、および display.settext 関数はオーバーラップ型の、他の動作をブロックしないタイプのコマンドです。つまり、スクリプトはこれらのコマンドの動作完了を待たずに次の動作を実行します。このような非ブロックタイプの関数は性能上の考慮から、ディスプレイを即時には更新せず、まず裏で処理を完了させておいてから処理時間に余裕が生ずるのを待って画面を更新します。
詳細	「カーソル位置」（p.14-5）参照。
参照	display.clear, display.getcursor, display.gettext, display.settext
例	カーソルを 2 行目の第 1 桁へ移動させます。 `display.setcursor(2, 1)`

display.settext	
機能	ユーザ画面にテキストを表示します。
使用法	`display.settext(text)` `text` 表示の対象となるテキスト文字列
備考	• この関数はユーザ表示画面を選択し、指定されたテキストを表示します。電源投入後の最初の書き込みによってユーザ画面がクリアされます。 • テキストは現在のカーソル位置を起点として表示され、テキスト表示後のカーソルは文字列の最後の文字の 1 つ後へ移動します。 • 上の行の表示テキストが行内に収まらなかったとしても下の行に自動的にラップされることはありません。現在の行に収まらなかった余分なテキストは切り捨てられます。テキストの最後の部分が切り捨てられた場合、カーソルは行の最後の桁に表示されます。 • テキストは書き換えまたはクリアされるまで画面上に留まります。 • テキスト文字列内は以下の文字コードを挿入することができます。 `$N` 改行 - この文字以後のテキストを次の行に表示します。カーソルがすでに 2 行目上にあった場合、‘$N’ 以後のテキストは無視されます。 `$R` テキストを通常（Normal）表示 `$B` テキストを点滅（Blink）表示 `$D` テキストを低輝度（Dim）表示 `$F` テキストのバックグラウンドを点滅表示 `$$` 1 個の “$” を表示するエスケープシーケンス • display.clear、display.setcursor、および display.settext 関数はオーバーラップ型の、他の動作をブロックしないタイプのコマンドです。つまり、スクリプトはこれらのコマンドの動作完了を待たずに次の動作を実行します。このような非ブロックタイプの関数は性能上の考慮から、ディスプレイを即時には更新せず、まず裏で処理を完了させておいてから処理時間に余裕が生ずるのを待って画面を更新します。
詳細	「テキストメッセージの表示」（p.14-6）参照。
参照	display.clear, display.getcursor, display.gettext, display.setcursor

例	ユーザ画面にメッセージを表示します。
	`display.clear()`
	`display.settext("Message Test $N$Bwith Row 2 Blinking")`
	上の行に “Message Test” が表示され、下の行には “with Row 2 Blinking” が点滅モードで表示されます。

display.smuX.digits	X = SMU チャンネル（a または b）
属性値	選択されている測定表示分解能。
使用法	`digits = display.smuX.digits` -- 分解能の読み取り `display.smuX.digits = digits` -- 分解能の書き込み 桁数に次のいずれかの値をセットします。 4 または display.DIGITS_4_5　4-1/2 桁分解能を選択 5 または display.DIGITS_5_5　5-1/2 桁分解能を選択 6 または display.DIGITS_6_5　6-1/2 桁分解能を選択
備考	• この属性値を指定することにより表示分解能を 4-1/2 桁、5-1/2 桁、6-1/2 桁のいずれかに設定します。 • SMU A と SMU B に別な表示分解能を設定することができます。
詳細	「表示分解能」（p.14-4）参照。
例	SMU A を 5-1/2 桁分解能に設定します。 `display.smua.digits = display.DIGITS_5_5`

display.smuX.measure.func	X = SMU チャンネル（a または b）
属性値	現在表示されている測定のタイプ。
使用法	`func = display.smuX.measure.func` -- 機能の読み取り `display.smuX.measure.func = func` -- 機能の書き込み func に次のいずれかの値をセットします。 0 または display.MEASURE_DCAMPS　電流測定機能を選択 1 または display.MEASURE_DCVOLTS　電圧測定機能を選択 2 または display.MEASURE_OHMS　抵抗測定機能を選択 3 または display.MEASURE_WATTS　電力測定機能を選択
備考	• 表示の対象となる測定機能を選択します：電流、電圧、抵抗、電力。 • SMU A と SMU B に別々な測定機能を設定することができます。
詳細	「測定機能」（p.14-3）参照。
例	SMU A の機能として電流測定を選択します。 `display.smua.measure.func = display.MEASURE_DCAMPS`

display.trigger.clear	
機能	前面パネルのトリガイベント検出回路をクリアします。
使用法	`display.trigger.clear()`
備考	トリガイベント検出回路は最後に display.trigger.wait を呼び出して以後に検出したイベントを記憶しています。この関数はトリガイベント検出回路をクリアして、それまでの TRIG キープレス履歴を破棄します。
詳細	「表示トリガ」(p.14-15) 参照。
参照	display.trigger.wait

display.trigger.wait	
機能	前面パネルの TRIG キーが押されるのを待ちます。
使用法	`triggered = display.trigger.wait(timeout)` `timeout` タイムアウト時間（秒）を設定します。 `triggered` トリガを検出すると true を返し、検出せずにタイムアウトになると false を返します。
備考	• この関数は前面パネルの TRIG キーが押されるのを待ちます。トリガキーが押された後で 1 つ以上のトリガイベントを検出すると、この関数は直ちにリターンします。 • この関数を使用してトリガ待ちを行った後は、イベントディテクタが自動的にリセットされて次回の準備が行われます。この動作は検出したイベントの数によらず同様です。 • トリガイベント検出回路をクリアするには display.trigger.clear を呼び出してください。
詳細	「表示トリガ」(p.14-15) 参照。
参照	display.trigger.clear

例	最高 5 秒間の間、TRIG キーが押されるのを待ちます。
	`triggered = display.trigger.wait(5)`
	`print(triggered)`
	Output: `true`
	上に示す出力例は、5 秒間のタイムアウト終了前に TRIG キーが押された（トリガ検出）ことを示します。

display.waitkey

機能

次回いずれかのキーが押されるのを待ち、そのキーコードを捕捉して返します。

使用法

```
key = display.waitkey()
```

備考

- この関数を送信すると、その後で前面パネルキーまたはホイールが押されるか、またはホイールが右 / 左に回されるまでスクリプトの実行が休止状態になります。いずれかのコントロール要素またはホイールが押されると、そのキーに対応するキーコードが返されます。前面パネルの個々のコントロール要素に対応するキーコードをまとめて下の表に示します。コントロール要素はアルファベット順に並べられています。
- この関数がキープレスの待ち状態にあるときに EXIT キーが押されたとしても、スクリプトは中断されません。
- この関数の標準的な使用目的は、プロンプトを表示してユーザに EXIT キー（スクリプト中断）または他のキー（スクリプト実行再開）を押させることです。キーコード値 75（EXIT キープレス）が返されたときに、exit() 関数を呼び出すことによりスクリプトをその時点で中断させることができます。この処理を実行するコード例が「キープレスコードの取り込み」（p.14-17）で説明されています。

コントロール要素	キーコード	コントロール要素	キーコード	コントロール要素	キーコード
AUTO	73	LIMIT (B)	90	REL (A)	70
CONFIG	80	LOAD	95	REL (B)	67
CURSOR (left)	104	MEAS (A)	86	RUN	71
CURSOR (right)	103	MEAS (B)	83	SPEED (A)	94
DIGITS (A)	87	MENU	68	SPEED (B)	91
DIGITS (B)	84	MODE (A)	69	SRC (A)	79
DISPLAY	72	MODE (B)	66	SRC (B)	76
ENTER	82	OUTPUT (A)	88	STORE	78
EXIT	75	OUTPUT (B)	96	TRIG	92
FILTER (A)	77	RANGE (down)	81	Wheel (press)	97
FILTER (B)	74	RANGE (up)	65	Wheel (left)	107
LIMIT (A)	93	RECALL	85	Wheel (right)	114

上の表は前面パネルのコントロール要素に対応するキーコード値をまとめたものです。それぞれのキーコード値に対応する識別コードが display.sendkey を説明した項にまとめられています（たとえば、RUN キーには display.KEY_RUN が対応します）。

詳細

「キープレスコードの取り込み」（p.14-17）参照。

参照

display.sendkey, display.settext, display.getlastkey

例

次のコードを実行すると、スクリプトの実行を一時停止してオペレータがキーまたはホイールを押すか、またはホイールを回転させるのを待ちます。

```
key = display.waitkey()
print(key)
```

Output: `8.600000e+01`

上に示した出力例（86）は MEAS(A) キーが押されたことを示しています。

errorqueue 関数と属性値

このグループに属する関数と属性値を使用してエラー / イベントキューに書き込まれている項目を読み出します。

errorqueue.clear	
機能	エラー / イベントキューに書かれている項目をすべて取り出してクリアします。
使用法	`errorqueue.clear()`
備考	この関数はエラー / イベントキュー内の全項目を取り除きます。
詳細	付録 B（エラー / ステータスメッセージ）、付録 D（ステータスモデル）参照。
参照	errorqueue.count, errorqueue.next

errorqueue.count	
属性値	エラー / イベントキュー内に存在する項目数。
使用法	`count = errorqueue.count`
備考	この属性値を読みだすことにより、エラー / イベントキュー内に存在するメッセージの数を知ることができます。この属性値は読み出し専用であり、値を書き込もうとするとエラーが発生します。
詳細	付録 B（エラー / ステータスメッセージ）、付録 D（ステータスモデル）参照。
参照	errorqueue.clear, errorqueue.next

例	エラー / イベントキュー内に存在する項目数を読み出します。
	`count = errorqueue.count`
	`print(count)`
	Output: `4.000000e+00`
	上の出力値から、イベント / エラーキューには全部で 4 項目が存在することが分ります。

errorqueue.next

機能	エラー / イベントキューに書き込まれている項目を 1 つ読み取ります。
使用法	`errorcode, message, severity, node = errorqueue.next()` `errorcode` 読み取った項目のエラー番号を返します。 `message` 項目を説明するメッセージを返します。 `severity` 重大度のレベルを返します（0、10、20、30、40）。 `node` エラーが発生したノードの番号を返します。
備考	• 項目は先入れ先出し（FIFO）形式のキューに保存されています。この関数は最も古い項目を読み出して、その項目をキューから削除します。 • エラーコードとメッセージ一覧についてはセクション B の表 B-2 をご覧ください。 • キューに何も書き込まれていない場合はコード 0（“Queue Is Empty”）を返します。 • 返される重大度のレベルは次の通りです。 0 通知 - エラーは発生していません。“Queue is Empty” 10 通知 - イベントまたは軽微なエラー。例：“Reading Available”、“Reading Overflow” 20 修復可能 - たとえば無効なユーザ入力。動作継続可能ですが、エラーは是正しなければなりません。例：“Exponent Too Large” 、“Numeric Data Not Allowed” 30 重大 - 重大なエラーが発生しており、技術サービスからの助言が必要かも知れません。例：“Saved calibration constants corrupted” 40 致命的 - 260x 型は動作を停止しており、修理が必要です。技術サービスの連絡先はセクション 1 に記載されています。例：“Bad SMU AFPGA image size”、“SMU is unresponsive” 、“Communication Timeout with DFPGA” • 拡張システム内の各 TSP-Link 対応装置にはノード番号が割り付けられています。node はエラーの発生元となったノード番号を返します。
詳細	付録 B（エラー / ステータスメッセージ）、付録 D（ステータスモデル）を参照。
参照	errorqueue.clear, errorqueue.count

例	エラー / イベントキューの最も古い項目を読み出します。
	`errorcode, message = errorqueue.next()`
	`print(errorcode, message)`
	Output: `0.000000e+00 Queue Is Empty`
	上記の出力例はキューが空であったことを示します。

exit 関数

現在実行中のスクリプトを終了させます。

exit	
機能	スクリプトの実行を停止させます。
使用法	`exit()`
備考	スクリプトの実行を強制的に中断します。オーバーラップして実行中のコマンドが存在していたとしても、その完了を待たずにスクリプトの実行を停止させます。オーバーラップしているコマンドを終了させる必要がある場合は exit 呼び出しの前に waitcomplete 関数を使用してください。このコマンドはスクリプト内でのみ機能します。

format 属性値

format 属性値を使用して printnumber および printbuffer 関数の出力形式を設定します。これらの属性値がデータ形式（ASCII またはバイナリ）、ASCII 精度（数値桁数）およびバイナリ形式でのバイト並び順序（順並び、逆並び）などを決定します。

format.asciiprecision	
属性値	ASCII 形式で出力する数値の精度（数値桁数）。
使用法	`precision = format.asciiprecision` -- 精度読み取り `format.asciiprecision = precision` -- 精度書き込み `precision` 1 ～ 16 の範囲で設定
備考	• この属性値は print、printnumber および printbuffer 関数が出力するデータの精度（数値桁数）を指定します。この精度属性値は ASCII 形式にのみ使用でき、精度を 1 から 16 までの数で指定します。 • 精度は出力される数値の有効数字桁数を表しています。少数点の左には必ず 1 桁以上の数が存在しますから、精度を指定する場合はこれらの桁を忘れないようにしてください。 • デフォルト精度（reset）は 6 です。
参照	format.byteorder, format.data, printbuffer, printnumber

例	ASCII 精度を 7 に設定して数値を出力します。
	`format.asciiprecision = 7`
	`print(2.5)`
	Output: 2.500000E+00

format.byteorder	
属性値	printnumber、printbuffer 関数が出力するデータの 2 進バイトオーダー。
使用法	`order = format.byteorder` -- バイトオーダー読み出し `format.byteorder = order` -- バイトオーダー書き込み order には次のいずれかの値をセットします。 0 または format.NORMAL 最上位バイト先頭 0 または format.BIGENDIAN 最上位バイト先頭 0 または format.NETWORK 最上位バイト先頭 1 または format.SWAPPED 最下位バイト先頭 1 または format.LITTLEENDIAN 最下位バイト先頭
備考	• この属性値は printnumber、printbuffer 関数を使用してデータを出力するときのバイトの並び順を指定します。この属性値を適用できるデータ形式は SREAL、REAL、REAL32、および REAL64 のみです。 • NORMAL、BIGENDIAN、および NETWORK は同じバイトオーダーを意味します。SWAPPED および LITTLEENDIAN は同じバイトオーダーを意味します。これらは同じ意味を持つ定数であり、ユーザの好みに応じてどちらを使用してもかまいません。 • IBM PC 互換機へデータを送るときは SWAPPED または LITTLEENDIAN バイトオーダーを選択してください。
参照	format.asciiprecision, format.data, printbuffer, printnumber
例	SWAPPED バイトオーダーを選択します。 `format.byteorder = format.SWAPPED`

format.data	
属性値	printnumber、printbuffer 関数を使用してデータを出力するときのデータ形式。
使用法	`fmt = format.data` -- データ形式読み取り `format.data = fmt` -- データ形式書き込み fmt には次のいずれかの値をセットします。 1 または format.ASCII ASCII 形式 2 または format.SREAL 単精度 IEEE-754 バイナリ形式 2 または format.REAL32 単精度 IEEE-754 バイナリ形式 3 または format.REAL 倍精度 IEEE-754 バイナリ形式 3 または format.REAL64 倍精度 IEEE-754 バイナリ形式
備考	• この属性値は printnumber および printbuffer 関数を使用してデータを出力するときのデータ形式を指定します。 • ASCII 形式の精度を指定するには format.asciiprecision 属性値を使用します。SREAL、REAL、REAL32、および REAL64 のバイトオーダーを指定するには format.byteorder 属性値を使用します。 • REAL32 と SREAL は同じ単精度形式を意味します。REAL と REAL64 は同じ倍精度形式を意味します。これらは同じ意味を持つ定数であり、ユーザの好みに応じてどちらを使用してもかまいません。 • IEEE-754 バイナリ形式は単精度の数値には 4 バイト、倍精度の数値には 8 バイトを使用します。 • バイナリのどの形式でデータを出力する場合も、応答メッセージは “#0” で始まり改行コードで終了します。ASCII 形式でデータを出力する場合は、各要素をコンマとスペース文字で区切って出力します。
参照	format.asciiprecision, format.byteorder, printbuffer, printnumber
例	データ形式として ASCII を選択します。 `format.data = format.ASCII`

gpib 属性値

次の属性値を使用して GPIB アドレスを設定します。

gpib.address	
属性値	GPIB アドレス
使用法	`address = gpib.address` -- アドレス読み出し `gpib.address = address` -- アドレス書き込み `address` 0 ～ 30 の範囲で設定
備考	• 新しい GPIB アドレスが有効になるのはコマンドが処理されるときです。コマンドが処理されるときに出力キュー内に応答メッセージが存在した場合、これらのメッセージを読みだすには新しいアドレスを使用しなければなりません。 • 装置の通信を再開する前に、コマンドが処理されるために充分な時間を置く必要があります。このコマンドを送信したならば、それ以後の装置との通信には必ず新しいアドレスを使用してください。 • GPIB アドレスは不揮発性メモリに保存されます。reset 関数はアドレス設定に影響を与えません。
詳細	セクション 11「GPIB の操作」を参照。

例	260x 型の GPIB アドレスを 26 に設定し、そのアドレスを読み込みます。
	`gpib.address = 26`
	`address = gpib.address`
	`print(address)`
	Output: `2.600000e+01`

localnode 属性値

このグループの属性値を使用して電源周波数の設定、ディスプレイ上のプロンプト（ON/OFF）およびエラーメッセージ（表示 / 非表示）をコントロールします。

localnode.linefreq	
属性値	電源周波数
使用法	`frequency = localnode.linefreq` -- 電源周波数読み取り `localnode.linefreq = frequency` -- 電源周波数書き込み `frequency` 50 または 60 に設定します。
備考	• 整数 NPLC アパーチャを使用する測定でのノイズ削減を実現するには、電源周波数設定が実際の AC 電源ラインの周波数（50/60Hz）と一致していなければなりません。 • 拡張システム（TSP-Link）で使用する場合、localnode.linefreq はリモートマスタノードにのみ送信されます。このコマンドを別な任意のノードへ送るには node[N].linefreq（N はノード番号）を使用してください。TSP- Link の詳しい説明についてはセクション 9 をご覧ください。
例	電源周波数を 60Hz に設定します。 `localnode.linefreq = 60`

localnode.prompts	
属性値	プロンプト表示モード
使用法	`prompting = localnode.prompts` -- プロンプトの状態読み取り `localnode.prompts = prompting` -- プロンプトの状態書き込み `prompting` 0（ディスエーブル）または 1（イネーブル）をセットします。
備考	• この属性値はプロンプト表示をコントロールします。値を 1 にセットしておくと、コマンドメッセージが装置によって処理される度にプロンプトが発行され、0 にセットしておくとプロンプトは表示されません。 • コマンドメッセージ自体がプロンプトを発生させるのではなく、260x 型がコマンドメッセージへの応答としてプロンプトを発生させます。 • プロンプトモードをイネーブルしておくと、260x 型はコマンドメッセージへ応答してプロンプトを発生させます。応答として返されるプロンプトには次の 3 種類があります。 • 標準プロンプト “TSP>”。このプロンプトはすべてが正常に機能しており、コマンドの処理が終了したことを示します。 • “TSP?” はプロンプト発行時のエラーキューが空ではないことを示します。“TSP>” と同様に、コマンドの処理は終了しています。その前に実行されたコマンドがエラーを発生したことを必ずしも意味せず、単にコマンド実行時にエラーキューが空ではなかったことを示します。 • “>>>>” は継続中を示すプロンプトであり、スクリプトやフラッシュイメージのダウンローディング中に表示されます。スクリプトやフラッシュイメージをダウンロードする時は、多数のコマンドメッセージが一まとめに送信されます。この継続プロンプトが表示されたときは、装置が現在のコマンドの一部としてさらに後続のメッセージ受信を期待していることを示します。 • Test Script Builder（TSB）はその動作にプロンプトを必要とするため、バックグラウンド動作でプロンプトモードを設定します。TSB はプロンプトの到着によってコマンドの実行終了を知るため、ユーザがプロンプトをディスエーブル設定すると TSB はハングアップしてしまいます。したがって、TSB を使用する場合は絶対にプロンプトをディスエーブルしないでください。 • 拡張システム（TSP-Link）で使用する場合、localnode.prompt はリモートマスタノードにのみ送信されます。このコマンドを別な任意のノードへ送るには node[N].prompt（N はノード番号）を使用してください。TSP- Link の詳しい説明についてはセクション 9 をご覧ください。
参照	localnode.showerrors
例	プロンプトをイネーブルします。 `localnode.prompts = 1`

localnode.showerrors	
属性値	エラーを自動的に表示。
使用法	`errormode = localnode.showerrors` -- エラー表示の状態取得 `localnode.showerrors = errormode` -- エラー表示の状態書き込み `errormode` 0 または 1 に設定します。
備考	• この属性値をエラー発生に対して 1 にセットしておくと、装置はエラーキューに登録されたエラー項目をすべて自動的に表示してエラーキューをクリアします。エラーはコマンドメッセージの実行が終了した時点で処理されます（プロンプトがイネーブルされている場合はプロンプト発行直前）。 • この属性値を 0 にセットしておくと、エラーはエラーキューに残されますから、明示的に読み込み / クリア処理を実行しなければなりません。 • 拡張システム（TSP-Link）で使用する場合、localnode.showerrors はリモートマスタノードにのみ送信されます。このコマンドを別な任意のノードへ送るには node[N].showerrors（N はノード番号）を使用してください。TSP- Link の詳しい説明についてはセクション 9 をご覧ください。
詳細	「errorqueue 関数と属性値」の項を参照。
参照	localnode.prompts
例	エラーを表示します。 `localnode.showerrors = 1`

makegetter 関数

これらの関数を使用して属性値を返送 / セットします。

makegetter	
機能	属性値を返す機能を持つ関数を作成します。
使用法	`getter = makegetter(table, attributename)` `table` 属性値が配置された読み出し専用テーブル。 `attributename` 属性値の名前を表す文字列。 `getter` 指定された属性の値を返す関数。

備考	• この関数を使用して、呼び出されたときに指定の属性値を返す関数を作成します。直接属性値へアクセスするよりも getter 関数呼び出しの方が一般に高速であるため、この関数を使用して属性値のエイリアスを作成すると実行効率を上げるために役立ちます。 • 属性値を何回も呼び出す必要がある場合は getter 関数を作成するのが効率的ですが、そうでない場合は getter 関数作成のためのオーバーヘッドの方が属性値の直接アクセスの時間よりも大きくなりますのであまり効果がありません。

例	getlevel という名前の getter 関数を作成します。
	`getlevel = makegetter(smua.source, "levelv")`
	`...`
	`v = getlevel()`
	getlevel を呼び出すと、この関数は smua.source.levelv の値を返します。

makesetter	
機能	属性値をセットする機能を持つ関数を作成します。
使用法	`setter = makesetter(table, attributename)` `table` 属性値が配置された読み出し専用テーブル。 `attributename` 属性値の名前を表す文字列。 `setter` 指定された属性に値を設定する関数。
備考	• この関数を使用して、呼び出されたときに指定の属性値に値を設定する関数を作成します。直接属性値へアクセスするよりも setter 関数呼び出しの方が一般に高速であるため、この関数で属性値のエイリアスを作成すると実行効率を上げるために役立ちます。 • 属性値を何回も呼び出す必要がある場合は setter 関数を作成するのが効率的ですが、そうでない場合は setter 関数作成のためのオーバーヘッドの方が属性値の直接アクセスの時間よりも大きくなりますのであまり効果がありません。

例	setlevel という名前の setter 関数を作成します。
	`setlevel = makesetter(smua.source, "levelv")`
	`for v = 1, 10 do`
	`  setlevel(v)`
	`end`
	ループ内で setlevel 関数を反復使用して smua.source.levelv をセットすることにより、ソーススイープを実行しています。

opc 関数

この関数は、オーバーラップするすべてのコマンドの実行が完了するとステータスレジスタの OPC ビットをセットします。

opc	
機能	オーバーラップするすべてのコマンド処理が完了すると OPC（Operation Complete）ステータスビットをセットします。
使用法	`opc()`
備考	• すでに起動されているローカルにオーバーラップするコマンドがすべて完了すると、この関数によって Standard Event Status Register の Operation Complete ビットがセットされます。各ノードがそれぞれ独立に自分のステータスモデルの Operation Complete ビットをセットすることに注意してください。 • オーバーラップコマンドをアクティブに実行していないノードがあれば、そのノードのビットは即時セットされます。それ以外のノードは自分のオーバーラップコマンドを完了させた時点でビットをセットします。
詳細	セクション D「ステータスモデル」を参照。
参照	waitcomplete

printbuffer 関数と printnumber 関数

これらの関数を使用してデータや数字を出力します。

printbuffer	
機能	読み込みバッファのサブテーブルからデータを出力します。

使用法	何個のサブテーブルを使用するかによって、この関数には何通りもの使用法があります。
	`printbuffer(start_index, end_index, st_1)`
	`printbuffer(start_index, end_index, st_1, st_2)`
	`printbuffer(start_index, end_index, st_1, st_2, ..., st_n)`
	`start_index` 出力する値の開始インデックス
	`end_index` 出力する値の終了インデックス
	`st_1, st_2, … st_n` 値の出力元となるサブテーブル

備考	• 現在実行中のオーバーラップコマンドが存在しない場合にデータを取得するための正しい使用法： `1 <= start_index <= end_index <= n` ここに、n は出力元のテーブルの最後の項目を表すインデックスを表します。 end_index < start_index または n < start_index の場合はデータは出力されません。start_index < 1 の場合は最初のインデックスとして 1 を用います。n < end_index の場合は最後のインデックスとして n を用います。 • 指定された読み込みバッファのいずれかがオーバーラップするコマンドによって使用されており、少なくとも希望するインデックスまで処理が終わっていない場合、この関数はデータが利用可能な状態になるのを待ってデータを返します。 • データ取得コマンドがオーバーラップしており、そのコマンドがまだ完了していない場合、n はその測定完了時にテーブルへ追加される最後のデータに対応するインデックスを指します。 • この関数は読み込みバッファからの値を出力します。特定のサブテーブルが指定されていない場合（たとえば、“rb1.statuses”ではなく “rb1”指定されているとき）、デフォルトの「読み込み」サブテーブルが使用されます。 • 少なくとも 1 つのサブテーブルが指定されている必要があります。出力書式によって規定される上限が存在し、出力メッセージの最大長も決められています。複数の値が 1 つのメッセージに織り込まれますから、出力メッセージの最大長を超えないように注意が必要です。 • すべてのデータが 1 つの応答メッセージにまとめて書き込まれ、この応答メッセージは format.data その他の関連属性値によって形式が規定されます。
参照	format.data, printnumber

例	次に示す例は 1 つのバッファ内のすべてのタイムスタンプと読み取り値、およびもう 1 つのバッファ（n=4）のすべての読み取り値を出力します。
	`format.data = format.ASCII`
	`printbuffer(1, rb1.n, rb1.timestamps, rb1, rb2)`
	返されるデータの例（timestamps、rb1.readings、rb2.readings）：
	`1.02345E-04, 8.76542E-04, 5.29372E-01, 1.02445E-04, 8.66543E-04, 5.24242E-01, 1.02545E-04, 8.56547E-04, 5.19756E-01, 1.02645E-04, 8.44546E-04, 5.14346E-01`

printnumber	
機能	読み込みバッファの出力用として選択された書式を用いて数字を出力します。

使用法	何個の数字を出力するかによって、この関数の使用法には何通りもの方法があります。
	`printnumber(v1)`
	`printnumber(v1 ,v2)`
	`printnumber(v1 ,v2, ..., vn)`
	`v1, v2, ..., vn` 出力の対象となる数字

備考	• この関数は与えられた数字を、format.data とその他の関連属性値で指定された書式を用いて出力します。 • 少なくとも 1 個の数字を与える必要があります。数字には出力書式によって規定される上限が存在し、出力メッセージの最大長も決められています。すべての値が 1 つのメッセージに書き込まれますから、出力メッセージの最大長を超えないように注意が必要です。
参照	printbuffer, format.data

例	以前実行した測定の 3 個の測定値を出力します。
	`format.data = format.ASCII`
	`printnumber(i, v, t)`
	返されるデータの例 (i, v, t):
	`1.02345E-04, 8.76542E-02, 5.29372E-01`

reset 関数

この関数はすべての論理装置の状態をデフォルト設定に戻します。

reset	
機能	論理装置をリセットしてデフォルト設定へ戻します。
使用法	`reset()`
備考	• この関数はシステム内のすべての論理装置をリセットします。システム内のすべての論理装置に順次アクセスして個々に reset メソッドを呼び出すことと等価です。 • デフォルト設定は表 1-3 に示されています。
詳細	セクション 1 の「デフォルト設定」参照。

serial 関数と属性値

このグループに属する関数と属性値を使用して RS-232 インターフェイスを設定します。

serial.baud	
属性値	RS-232 ポートのボーレート
使用法	`baud = serial.baud` -- ボーレート読み込み `serial.baud = baud` -- ボーレート書き込み `baud` 次の何れかの値に設定：300、600、1200、2400、4800、9600、19200、38400、57600、115200
備考	• 新しいボーレート設定は変更コマンドの処理中に有効となります。 • 変更コマンドが処理されるために充分な時間を置いてから装置の通信を再開する必要があります。ボーレート設定は GPIB インターフェイスまたは前面パネルから行うようにお奨めします。 • ボーレートは不揮発性メモリに保存されます。reset 関数はボーレート設定に影響を与えません。
詳細	セクション 11「RS-232 インターフェイスの操作」を参照。
参照	serial.databits, serial.flowcontrol, serial.parity
例	ボーレートを 1200 に設定します。 `serial.baud = 1200`

serial.databits	
属性値	RS-232 ポートが使用するキャラクタ幅（データビット）。
使用法	`bits = serial.databits` -- データ幅読み取り `serial.databits = bits` -- データ幅書き込み `bits` 7 または 8 に設定します。
備考	• 新しいデータ幅は変更コマンドの処理中に有効となります。 • 変更コマンドが処理されるために充分な時間を置いてから装置の通信を再開する必要があります。データ幅設定は GPIB インターフェイスまたは前面パネルから行うようにお奨めします。 • データビット値は不揮発性メモリに保存されます。reset 関数はデータビットに影響を与えません。
詳細	セクション 11「RS-232 インターフェイスの操作」を参照。
参照	serial.baud, serial.flowcontrol, serial.parity
例	データ幅を 8 に設定します。 `serial.databits = 8`

serial.flowcontrol	
属性値	RS-232 ポートのフローコントロール。
使用法	`flow = serial.flowcontrol` -- フローコントロールの設定読み取り `serial.flowcontrol = flow` -- フローコントロール設定書き込み flow に次のいずれかの値をセットします。 "none" または serial.FLOW_NONE フローコントロールを使用しません "hardware" または serial.FLOW_HARDWARE ハードウェアによるフローコントロールを選択
備考	• 新しいフローコントロール設定は変更コマンドの処理中に有効となります。 • 変更コマンドが処理されるために充分な時間を置いてから装置の通信を再開する必要があります。フローコントロール設定は GPIB インターフェイスまたは前面パネルから行うようにお奨めします。 • フローコントロール設定値は不揮発性メモリに保存されます。reset 関数はフローコントロールに影響を与えません。
詳細	セクション 11「RS-232 インターフェイスの操作」を参照。
参照	serial.baud, serial.databits, serial.parity
例	フローコントロールを “none”（不使用）に設定します。 `serial.flowcontrol = serial.FLOW_NONE`

serial.parity	
属性値	RS-232 ポートのパリティ。
使用法	`parity = serial.parity` -- パリティ読み込み `serial.parity = parity` -- パリティ書き込み parity に次のいずれかの値をセットします。 "none" または serial.PARITY_NONE　パリティを使用しません "even" または serial.PARITY_EVEN　偶数パリティ選択 "odd" または serial.PARITY_ODD　奇数パリティ選択
備考	• 新しいパリティ設定は変更コマンドの処理中に有効となります。 • 変更コマンドが処理されるために充分な時間を置いてから装置の通信を再開する必要があります。パリティ設定は GPIB インターフェイスまたは前面パネルから行うようにお奨めします。 • パリティ設定は不揮発性メモリに保存されます。reset 関数はパリティに影響を与えません。
詳細	セクション 11「RS-232 インターフェイスの操作」を参照。
参照	serial.baud, serial.databits, serial.flowcontrol
例	パリティを不使用に設定します。 `serial.parity = serial.PARITY_NONE`

serial.read	
機能	シリアルポートからデータを読み込みます。
使用法	`data = serial.read(maxchars)` `maxchars`　読み込む文字数の最大値をセットします。 `data`　シリアルポートから読み込んだ全部のデータを含む文字列を返します。
備考	• シリアルポートにすでに存在するキャラクタを読み込むのがこの関数の機能であり、新しいキャラクタの受信待ちは行いません。通常は maxchars に比較的小さい値（数百文字以下）を設定しておき、シリアルポートが関数呼び出し前に受信していたキャラクタを返すようにします。多くの場合は受信キャラクタ数が maxchars を下回りますが、この関数呼び出しの間に受信したキャラクタ数が非常多いときは RS-232 バッファがオーバーフローを起こして一部のキャラクタが失われることがあります。 • 必要な数のキャラクタを受け取るために、この関数は必要に応じて何回呼び出してもかまいません。ただし、性能上の観点からは関数の反復呼び出しの間に多少の遅延時間を置くようにお奨めします。 • 返されるデータはポートが受け取った生のデータストリームです。したがって、制御文字や終端文字等は処理されておらず、データストリームは全く何の変更も受けていません。 • シリアルポートがコマンドインターフェイスとして稼動している時はこの関数を使用できません。シリアルポートがコマンドインターフェイスとしてイネーブルされている状態でこの関数を呼び出すと設定上の矛盾によりエラーが生じます。

参照	serial.write

例	シリアルポートからデータを読み込みます。
	`data = serial.read(200)`
	`print(data)`
	Output: `John Doe`
	上記出力例から分る通り、シリアルポートからは文字列 “John Doe” が読み込まれました。

serial.write	
機能	シリアルポートにデータを書き込みます。
使用法	`serial.write(data)` `data` 書き込むデータ文字列を指定します。
備考	この関数は与えられた文字列をシリアルポートに書き込み、シリアルポートの他端に接続された装置（例：コンポーネントハンドラ）がそのデータを読取れるようにします。この関数は与えられたままのデータを書き込み、終端文字を追加する等の操作は一切行いません。
参照	serial.read
例	シリアルポートにデータ文字列 “1 2 3 4” を書き込みます。 `serial.write("1 2 3 4")`

setup 関数と属性値

このグループの関数と属性値を使用してセットアップの保存と読み出し、および装置起動時に使用するセットアップを指定します。

setup.poweron	
属性値	装置に電源が入ったときに呼び出す保存セットアップ。
使用法	`n = setup.poweron` -- 起動時セットアップの読み込み `setup.poweron = n` -- 起動時セットアップの書き込み `n` 装置起動時に呼び出すセットアップの番号（0 - 5）
備考	n に 0 を指定すると、装置は工場デフォルト設定（リセット時の設定）を使用して起動します。1 から 5 をセットしておくと、装置はユーザが保存していたセットアップを呼び出して立上がります。
詳細	セクション 1「リモート操作用のセットアップ」を参照。
例	装置が工場設定のデフォルト条件で起動するように設定します。 `setup.poweron = 0`

setup.recall	
機能	保存されているセットアップの中から選択して設定を呼び出します。
使用法	`setup.recall(n)` `n` 呼び出すセットアップの番号（0 - 5）。
備考	n に 0 を指定すると、装置は工場デフォルト設定（リセット時の設定）を呼び出します。1 から 5 をセットしておくと、装置はユーザが保存していたセットアップを呼び出します。
詳細	セクション 1「リモート操作用のセットアップ」を参照。
例	2 番目の位置に保存されているユーザセットアップを呼び出します。 `setup.recall(2)`

setup.save	
機能	現在のセットアップをユーザセットアップの 1 つとして保存します。
使用法	`setup.save(n)` `n` 保存するセットアップ番号（1 - 5）。
備考	ユーザセットアップの保存位置を示す番号として 1 から 5 までを使用します。すでに存在するセットアップと同じ番号を使用すると、古いセットアップは上書きされます。
詳細	セクション 1「リモート操作用のセットアップ」を参照。
例	現在のセットアップを位置番号 5 に保存します。 `setup.save(5)`

smuX 関数と属性値

このグループに属する関数と属性値を使用して SMU の基本ソースメジャー操作コントロールと装置校正を行います。

smuX.cal.date X = SMU チャンネル（a または b）	
属性値	現在アクティブな校正セットの校正実施日付。
使用法	`caldate = smuX.cal.date` -- 校正日付読み取り `smuX.cal.date = caldate` -- 校正日付書き込み caldate には以下の値を設定します。 `os.time({year=yr, month=mo, day=da})` ここに、`yr` 2005 to 2037 `mo` 1 to 12 `da` 1 to 31

備考	• この属性値は現在アクティブな校正セットの校正実施日付を保存します。校正日付は任意のタイミングで自由に読み出し可能ですが、smuX.cal.unlock 関数を使用して校正作業を許可したとき以外は書き込みが許されません。 • この属性値はアクティブな校正セットと共に保存されます。別な校正セットを呼び戻した場合、この属性値は呼び戻した校正セットの日付を反映します。 • 次の書式を用いれば、caldate に時間と分情報も含めることができます。 `os.time({year=yr, month=mo, day=da, hour=hr, minute=mn})` 同様に秒情報を含めることも可能ですが、内部の日付データ保存形式の精度上の制約のため基本的には無視されます。 • 日付は UTC（協定世界時間）1970 年 1 月 1 日 12:00 am を起点とする秒数として保存されます。
詳細	セクション 17（校正）参照。
参照	smuX.cal.due, smuX.cal.restore, smuX.cal.save, smuX.makebuffer
例	SMU A の校正日付（2005 年 7 月 1 日）を設定します。 `smua.cal.date = os.time({year=2005, month=7, day=1})`

smuX.cal.due

X = SMU チャンネル（a または b）

属性値	次回の校正実施予定日。
使用法	`caldue = smuX.cal.due` -- 校正予定日付読み取り `smuX.cal.due = caldue` -- 校正予定日付書き込み caldue には以下の値を設定します。 `os.time({year=yr, month=mo, day=da})` ここに、`yr` 2005 to 2037 `mo` 1 to 12 `da` 1 to 31
備考	• この属性値は現在アクティブな校正セットの次に行う校正実施予定日付を保存します。校正予定日付は任意のタイミングで自由に読み出し可能ですが、smuX.cal.unlock 関数を使用して校正作業を許可したとき以外は新しい値を書き込むことができません。 • この属性値はアクティブな校正セットと共に保存されます。別な校正セットを呼び戻した場合、この属性値は呼び戻した校正セットに付随した予定日を反映します。 • 次の書式を用いれば、caldate に時間と分情報も含めることができます。 `os.time({year=yr, month=mo, day=da, hour=hr, minute=mn})` 同様に秒情報を含めることも可能ですが、内部の日付データ保存形式の精度上の制約のため基本的には無視されます。 • 日付は UTC（協定世界時間）1970 年 1 月 1 日 12:00 am を起点とする秒数として保存されます。
詳細	セクション 17（校正）参照。
参照	smuX.cal.date, smuX.cal.restore, smuX.cal.state, smuX.makebuffer
例	SMU A の校正予定日付（2006 年 7 月 1 日）を設定します。 `smua.cal.due = os.time({year=2005, month=7, day=1})`

smuX.cal.lock	X = SMU チャンネル（a または b）
機能	校正設定変更コマンドの動作を禁止します。
使用法	`smuX.cal.lock()`
備考	この関数を実行すると、校正の設定変更に関わる校正関数が機能しなくなります。したがって、校正状態が smuX.CALSTATE_CALIBRATING のときにこの関数を呼び出すとエラーになります。校正定数は不揮発性メモリに書き込まなければなりません。または、校正をロックする前に以前の校正セットを呼び戻さなければなりません。
詳細	セクション 17（校正）参照。
参照	smuX.cal.restore, smuX.cal.save, smuX.cal.state, smuX.makebuffer
例	SMU A の校正関数を実行できなくします。 `smua.cal.lock()`

smuX.cal.password	X = SMU チャンネル（a または b）
属性値	校正の実行に必要なパスワード。
使用法	`smuX.cal.password = newpassword` `newpassword` 新規パスワード（文字列）。
備考	• 新規パスワードの割り付けが許されるのは校正がロックされていないときだけです。 • 校正パスワードの書き込みはできますが、読み出しはできません。
詳細	セクション 17（校正）参照。
参照	smuX.makebuffer
例	SMU A 用に新しい校正パスワードを割り付けます。 `smua.cal.password = "LetMeIn"`

smuX.cal.polarity	X = SMU チャンネル（a または b）
属性値	以後の測定でどちらの校正定数を使用するのかを指定します。
使用法	`calpolarity = smuX.cal.polarity` -- 校正の極性指定読み込み `smuX.cal.polarity = calpolarity` -- 校正の極性指定書き込み calpolarity に次のいずれかの値をセットします。 0 または smuX.CAL_AUTO 自動極性検出 1 または smuX.CAL_POSITIVE 正極性の校正定数を使用して測定します 2 または smuX.CAL_NEGATIVE 負極性の校正定数を使用して測定します

備考	• この属性値は、以後のすべての測定でどちらの極性の校正定数を使用するのかを指定します。smuX.measure.calibrateY / smuX.source.calibrateY 関数はこの属性値の影響を受けません。これらのコマンドの極性はそのコマンドに与えられるレンジパラメータによって規定されます。 • 測定校正コマンドを使用するためには、校正された極性を使用して取得された測定値を与える必要があります。校正ポイントがゼロから充分に離れていれば、これらの測定を行うときに当然ながら希望する極性定数が使用されますが、測定ポイントが非常にゼロに近い場合には測定にどちらの極性の校正定数が使われたのか判然としないことがあります。この属性値でどちらかの極性を明示的に指定することによって、生の測定データから判断するのではなく、指定された極性の校正定数を用いて測定が行われます。 • この属性値を正負何れかの極性に指定できるのは校正がロックされていないときだけです。校正がロックされると自動的に属性値が CAL_AUTO に戻されます。
詳細	セクション 17（校正）参照。
参照	smuX.makebuffer, smuX.measure.calibrateY, smuX.source.calibrateY
例	正極性の校正定数を以後のすべての測定に適用します。 `smua.cal.polarity = smua.CAL_POSITIVE`

smuX.cal.restore	X = SMU チャンネル（a または b）
機能	保存されている校正定数セットの 1 つをローティングします。

使用法	この関数は次の 2 通りの方法で使用します。
	`smuX.cal.restore()`
	`smuX.cal.restore(calset)`
	`calset` ローディングする校正セット。
	calset には次のいずれかの値をセットします。
	`smuX.CALSET_NOMINAL` 個々の装置特性に即した校正がなされていない、公称値を用いた校正定数セットですが、装置の基本的な動作には使用できます。
	`smuX.CALSET_FACTORY` 工場出荷時に装置に書き込まれる校正定数。
	`smuX.CALSET_DEFAULT` 通常の校正セット。
	`smuX.CALSET_PREVIOUS` デフォルトセットが最後に上書きされる前に使用されていた校正セット。
	calset に何も指定しなかった場合は smuX.CALSET_DEFAULT が使用されます。

備考	• この関数は、不揮発性メモリから呼び出した校正セットで現在の校正セットを上書きします。 • smuX.cal.unlock が成功裏に呼び出されるまでこの関数は機能しません。
詳細	セクション 17（校正）参照。
参照	smuX.makebuffer
例	SMU A の工場校正セットを呼び戻します。 `smua.cal.restore(smua.CALSET_FACTORY)`

smuX.cal.save	X = SMU チャンネル（a または b）
機能	校正定数を不揮発性メモリに保存します。
使用法	`smuX.cal.save()`
備考	• この関数は、現在の校正定数セットを不揮発性メモリに保存します。それ以前の校正定数（デフォルト校正セットからの定数）は、デフォルト校正セットを上書きする前に、以前の校正セット（CALSET_PREVIOUS）にコピーされます。 • この関数は smuX.cal.unlock が成功裏に呼び出されるまでは機能しません。また、校正定数のいずれかの項目が変更されている場合には、校正日付と次回の校正予定日付に新しい値が割り付けられるまで、この関数は機能できません。
詳細	セクション 17（校正）参照。
参照	smuX.cal.date, smuX.cal.due, smuX.cal.restore, smuX.makebuffer
例	SMU A が使用する校正定数を不揮発性メモリに保存します。 `smua.cal.save()`

smuX.cal.state	X = SMU チャンネル（a または b）
属性値	校正状態。
使用法	`calstate = smuX.cal.state`
備考	この読み取り専用属性値を読み取ると、calstate には次のいずれかの値が返されます。 0 `smuX.CALSTATE_LOCKED` 校正はロック（禁止）されています。 1 `smuX.CALSTATE_CALIBRATING` 校正定数または日付が変更されましたが、まだ不揮発性メモリに保存されていません。 2 `smuX.CALSTATE_UNLOCKED` 校正はロックされていません。かつ、いずれの校正定数や日付も前回の保存 / 呼び出し後変更されていません。
詳細	セクション 17（校正）参照。
参照	smuX.cal.due, smuX.cal.restore, smuX.cal.save, smuX.makebuffer

例	SMU A の校正状態を読み取ります。
	`calstate = smua.cal.state`
	`print(calstate)`
	Output: `0.000000e+00`
	上の出力例は、校正がロックされていることを示しています。

smuX.cal.unlock	X = SMU チャンネル（a または b）
機能	校正設定変更コマンドの動作を許可します。
使用法	`smuX.cal.unlock(password)` `password` 校正パスワード
備考	• この関数は校正関数に校正設定の変更を許します。 • 装置の工場出荷時には “KI002602”というパスワードが設定されています。
詳細	セクション 17（校正）参照。
参照	smuX.cal.password
例	SMU A 校正のロック（禁止）を解除します。 `smua.cal.unlock("KI002602")`

smuX.makebuffer	X = SMU チャンネル（a または b）
機能	RAM バッファを作成します。
使用法	`mybuffer = smuX.makebuffer(buffersize)` `buffersize` 保存可能な読み取り値の数。
備考	• RAM 読み込みバッファは動的割り付けが可能です。その作成と割り付けに使用するのが smuX.makebuffer(buffer) 関数であり、ここに buffersize はバッファに保存可能な読み取り値の数を表します。 • 動的に割り付けられた読み込みバッファも smuX.nvbufferY バッファと同等に扱うことができます。 • RAM バッファの削除には nil を使用します。たとえば、次のコマンドは mybuffer を削除します。 `mybuffer = nil`
詳細	セクション 7「読み込みバッファ」参照。
参照	smuX.nvbufferY
例	SMUA 用に 200 データを保存する RAM バッファを “mybuffer2”という名前で作成します。 `mybuffer2 = smua.makebuffer(200)`

smuX.measure.autorangeY	X = SMU チャンネル（a または b） Y = SMU 測定関数（v または i）：v= 電圧、i= 電流
属性値	測定の自動レンジ設定。

使用法	`autorange = smuX.measure.autorangeY` -- 測定自動レンジ読み取り `smuX.measure.autorangeY = autorange` -- 測定自動レンジ書き込み autorange に次のいずれかの値をセットします。 0 または smuX.AUTORANGE_OFF　測定自動レンジをディスエーブル 1 または smuX.AUTORANGE_ON　測定自動レンジをイネーブル
備考	• この属性値は測定自動レンジの状態を示します。SMU 測定回路が固定レンジに設定されていると smuX.AUTORANGE_OFF、自動レンジモードに入っている場合は smuX.AUTORANGE_ON という値を示します。 • この属性値を smuX.AUTORANGE_OFF にセットすると SMU は固定レンジモードに入り、その時点で SMU 測定回路が使用していたレンジが固定レンジとなります。 • この属性値を smuX.AUTORANGE_ON にセットすると SMU 測定回路が自動レンジモードに入ますが、次回の測定要求を受け付けるまでは現状の測定レンジに留まります。
詳細	セクション 6「レンジ」参照。
参照	smuX.measure.rangeY
例	SMU A 電圧測定の自動レンジ動作をイネーブルします。 `smua.measure.autorangev = smua.AUTORANGE_ON`

smuX.measure.autozero	X = SMU チャンネル（a または b）
属性値	SMU A/D 内部基準測定の挙動（オートゼロ）を示します。
使用法	`azmode = smuX.measure.autozero` -- オートゼロの状態読み取り `smuX.measure.autozero = azmode` -- オートゼロの状態書き込み azmode に次のいずれかの値をセットします。 0 または smuX.AUTOZERO_OFF　オートゼロ禁止。 1 または smuX.AUTOZERO_ONCE　オートゼロを 1 回実行し、その後はオートゼロ禁止とします。 2 または smuX.AUTOZERO_AUTO　基準値とゼロの測定を自動的にチェックします。必要に応じてオートゼロを実行します。

備考	• 260x 型ソースメータはレシオメトリック方式の A/D 変換を使用します。この方式は読み取り値の確度維持のために、定期的に内部 GND と基準電圧の値を取り込みます。これらの基準値測定が必要となる時間間隔は、その時点で測定に使用されている積分アパーチャの値に依存します。それぞれのアパーチャごとに基準値とゼロ測定が別々に行われます。 • デフォルト設定では、信号測定を行うときは必ず装置がこれらの基準値を自動的にチェックします。信号測定を実行したときにすでに基準値測定の有効期限が切れていた場合、装置は自動的に 2 回の A/D 変換（基準値とゼロ）を行います。このため、測定時間が標準よりも長くなることが時々あります。測定のタイミングが重要であるアプリケーションではこの余分な変換時間がスイープその他の試験シーケンスに支障を及ぼす可能性もあります。基準値測定に関わる余分な測定時間の発生を避けたい場合は、smuX.measure.autozero 属性値を利用して基準値の自動測定を禁止してください。ただし、基準値の自動測定を使用しないと装置が徐々にドリフトして仕様の精度から外れて行く可能性があるので注意が必要です。ドリフトを最小限に抑えるためには、少なくとも重要な試験シーケンスの実施前には必ず基準電圧とゼロの測定を行うべきです。現在のアパーチャ設定で基準電圧とゼロの測定値を強制的に更新させたい場合は smuX.AUTOZERO_ONCE 設定を使用してください。 • 最近使用された 5 種類の NPLC 設定でのオートゼロ基準測定値が基準値キャッシュに保存されています。ある NPLC 設定を選択したときに該当する項目がキャッシュに見つからない場合は、時間的に最も古い項目を破棄して新しい項目用のスペースを確保します。
例	SMU A のオートゼロを 1 回実行します。 `smua.measure.autozero = smua.AUTOZERO_ONCE`

smuX.measure.calibrateY	X = SMU チャンネル（a または b） Y = SMU 測定関数（v または i）：v= 電圧、i= 電流
機能	新しい測定校正定数を発生させ、それを有効にします。
使用法	`smuX.measure.calibrateY(range, cp1measured, cp1reference, cp2measured, cp2reference)` `range` 校正の対象となる測定レンジ `cp1measured` SMU が校正ポイント 1 から取得した測定値 `cp1reference` 校正ポイント 1 を外部で測定した基準となる測定値 `cp2measured` SMU が校正ポイント 2 から取得した測定値 `cp2reference` 校正ポイント 2 を外部で測定した基準となる測定値

備考	• この関数は与えられたレンジで使用する校正定数を新たに作成して、それらの値を有効にします。装置は正極性と負極性で別々に校正を行わなければなりません。あるレンジを正極性で校正する場合は正の値を用い、負極正で校正するには負の値を使用します。 • 2 つの校正ポイントを使用するのが標準的で、校正ポイント 1 をゼロ近傍、校正ポイントをフルスケールの 90% に設定します。 • 4 個の測定値（cp1measured、cp1reference、cp2measured、cp2reference）は必ず現在アクティブな校正セットを使用して取得しなければなりません。この規則が守られない場合は校正定数の意味が失われます。 • 新しい校正定数は作成後直ちに有効となりますが、そのまま不揮発性メモリに保存される訳ではありません。smuX.cal.save を使用して新しい定数を不揮発性メモリへ保存してください。現在アクティブな校正定数は、装置の電源を切って再投入するか、もしくは smuX.cal.restore 関数を使用して不揮発性メモリから別な校正セットを呼び出すまで有効です。 • smuX.cal.unlock が成功裏に呼び出されるまでこの関数は機能しません。
詳細	セクション 17（校正）参照。
参照	smuX.cal.restore, smuX.cal.save, smuX.makebuffer, smuX.source.calibrateY

smuX.measure.count

X = SMU チャンネル（a または b）

属性値	測定が要求されたときに実行される測定回数。
使用法	`count = smuX.measure.count` -- 測定回数読み込み `smuX.measure.count = count` -- 測定回数書き込み `count` 測定回数
備考	• この属性値は、測定要求が発生したときに実際に行われる測定回数を指定します。測定コマンドとともに読み込みバッファを使用する場合は、count は保存する読み込み値の数も同時に指定します。 • reset 関数は測定回数を 1 にリセットします。
詳細	セクション 4「トリガ」参照。
参照	smuX.measure.overlappedY, smuX.measure.Y
例	SMU A の測定回数を設定します。 `smua.measure.count = 10`

smuX.measure.filter.count X = SMU チャンネル（a または b）	
属性値	1 個のフィルタ出力計算に使用する読み取り値の数。
使用法	`count = smuX.measure.filter.count` -- フィルタカウント読み込み `smuX.measure.filter.count = count` -- フィルタカウント書き込み `count` フィルタカウントを 1 から 100 までの範囲で設定します。
備考	• この属性値は 1 個のフィルタ出力を計算するために使用する測定値の数を表します。 • reset 関数はフィルタカウントを 1 にリセットします。
詳細	セクション 6「フィルタ」参照。
参照	smuX.measure.filter.enable, smuX.measure.filter.type
例	SMU A のフィルタカウントを設定します。 `smua.measure.filter.count = 10`

smuX.measure.filter.enable X = SMU チャンネル（a または b）	
属性値	測定値のフィルタ処理の ON/OFF を設定します。
使用法	`filter = smuX.measure.filter.enable` -- フィルタ処理の ON/OFF 状態読み取り `smuX.measure.filter.enable = filter` -- フィルタ処理の ON/OFF 書き込み filter に次のいずれかの値をセットします。 0 または smuX.FILTER_OFF フィルタ処理を使用しません。 1 または smuX.FILTER_ON フィルタ処理を使用します。
備考	• この属性値はフィルタ処理使用の有無を指定します。 • reset 関数はフィルタ処理を不使用として設定します。
詳細	セクション 6「フィルタ」参照。
参照	smuX.measure.filter.count, smuX.measure.filter.type
例	SMU A がフィルタ処理を使用するように設定します。 `smua.measure.filter.enable = smua.FILTER_ON`

smuX.measure.filter.type	X = SMU チャンネル（a または b）
属性値	測定で使用するフィルタのタイプ。
使用法	`type = smuX.measure.filter.type` -- フィルタタイプ読み込み `smuX.measure.filter.type = type` -- フィルタタイプ書き込み type に次のいずれかの値をセットします。 0 または smuX.FILTER_MOVING_AVG 移動平均フィルタを選択します。 1 または smuX.FILTER_REPEAT_AVG 反復フィルタを選択します。
備考	• 平均化フィルタの形式には反復と移動の 2 つのタイプがあります。反復フィルタ（電源投入時のデフォルト設定）はスタックに読み取り値を積み上げてゆき（スタックの深さ＝ filter count）、それらの値の平均値を指示値とします。その後スタックを一度空にしてから次回の処理を開始します。移動平均フィルタは先入れ先出し（“first-in, first-out”）スタックを使用します。スタック（深さ＝ filter count）がいっぱいになると読み取り値を平均化して 1 つの指示値を計算します。それ以後は新しい読み取り値が得られる度にスタックにそれを追加すると同時に、一番古い読み取り値を捨て、スタックの平均値を再計算してそれを新しい指示値とします。 • reset 関数は反復フィルタを選択します。
詳細	セクション 6「フィルタ」参照。
参照	smuX.measure.filter.count, smuX.measure.filter.enable
例	SMU A のフィルタとして移動平均タイプを選択します。 `smua.measure.filter.type = smua.FILTER_MOVING_AVG`

smuX.measure.interval	X = SMU チャンネル（a または b）
属性値	複数測定相互の時間間隔。
使用法	`interval = smuX.measure.interval` -- 測定間隔読み取り `smuX.measure.interval = interval` -- 測定間隔書き込み `interval` 測定時間間隔を 0 から 1（秒）の範囲で設定します。
備考	• smua.measure.count に 1 を超える値を設定した場合、この属性値によって測定相互の時間間隔を指定します。SMU は各グループの測定が予定された時間に始まるように出来る限りの調整を行います。 • 測定値をフィルタ処理している場合、この属性値はフィルタ処理される 1 つのグループの最初の測定から次のグループの最初の測定までの時間を意味します。フィルタ処理のために必要な複数の測定の時間間隔を規定するものではありません。 • SMU がこの時間間隔を正確に守れない状況が発生した場合は、可能な限り速やかに測定を実行します。 • reset 関数は測定時間間隔を 0 にリセットします。
詳細	セクション 4「トリガ」参照。
参照	smuX.measure.overlappedY, smuX.measure.Y
例	SMU A の測定時間間隔を設定します。 `smua.measure.interval = 0.5`

smuX.measure.lowrangeY	X = SMU チャンネル（a または b） Y = SMU 測定関数（v または i）：v= 電圧、i= 電流
属性値	自動レンジ機能に選択を許す最低測定レンジ。
使用法	`rangeval = smuX.measure.lowrangeY` -- 最低レンジ読み取り `smuX.measure.lowrangeY = rangeval` -- 最低レンジ書き込み `rangeval` 電圧または電流の最低測定レンジをセットします。
備考	• この属性値を使用して自動レンジが選択可能な最低レンジを指定します。低いレンジの方が一般に長い整定時間を必要とします。レンジ値として大きな値を設定すると、より短い整定時間で測定が可能になることがあります。 • 装置を自動レンジに設定したときに、ここで指定する最低レンジよりも低いレンジに入っていた場合は自動的に指定の最低レンジに切り換わります。
詳細	セクション 6「レンジ」参照。
参照	smuX.measure.autorangeY
例	SMU A による電圧測定の低域レンジを 1V に設定します。 `smua.measure.lowrangev = 1`

smuX.measure.nplc	X = SMU チャンネル（a または b）
属性値	測定に使用する積分時間。
使用法	`nplc = smuX.measure.nplc` -- nplc 読み取り `smuX.measure.nplc = nplc` -- nplc 書き込み `nplc` 0.001 ～ 25 の範囲で設定します。
備考	• 積分時間は電源周波数から導かれるパラメータ NPLC(Number of Power Line Cycles) を用いて指定します。すなわち、1 PLC は 60Hz ならば 16.67msec（1/60）となり、50Hz ならば 20msec（1/50）となります。 • reset 関数は時間を 1.0 にリセットします。
詳細	セクション 6「速度」参照。
例	SMU A の積分時間を設定します（0.5/60 秒）。 `smua.measure.nplc = 0.5`

smuX.measure.overlappedY **smuX.measure.overlappediv**	X = SMU チャンネル（a または b） Y = SMU 測定関数（v、i、r 、p）： v = 電圧、i = 電流、r = 抵抗、p = 電力
機能	非同期（バックグラウンド）測定を開始します。

使用法	この関数は次の 2 通りの方法で使用します。
	`smuX.measure.overlappedY(rbuffer)`
	`smuX.measure.overlappediv(ibuffer, vbuffer)`
	`rbuffer` 読み取り値を保存する読み取りバッファオブジェクト
	`ibuffer` 電流読み取り値を保存する読み取りバッファオブジェクト
	`vbuffer` 電圧読み取り値を保存する読み取りバッファオブジェクト

備考	• この関数は測定を開始して直ちにリターンします。測定値は測定の実行ごとに読み込みバッファに書き込まれてゆきます（補足情報があるときはそれも同様に書き込まれてゆきます）。装置が複数の読み取り値を返すように設定されており、要求されているのがその中の 1 つであったとすれば、その読み取り値については取得後直ちに使用が可能です。 • smuX.measure.overlappediv 関数は電流と電圧両方の読み取り値をそれぞれのバッファに保存します（電流値、続いて電圧値が別々なバッファに保存されます）。 • この関数はオーバーラップコマンドです。すなわち、測定をバックグラウンドで実行しながらスクリプトが進行します。ただし、まだ完全に得られていない測定結果にアクセスしようとすると、そのデータが利用可能になるまでスクリプトの進行がブロックされます。測定が完了するのを待ってから次へ進めたい場合は waitcomplete 関数の使用が便利です。 • 指定された読み込みバッファ内に何らかのデータが残っている場合は、そのバッファが追加書き込み（アペンド）モードに設定されていない限り、測定実行前にクリアされます。
詳細	セクション 7「読み込みバッファ」参照。
参照	smuX.nvbufferY, smuX.nvbufferY, waitcomplete
例	SMU A の電圧測定をバックグランドで開始します。 `smua.measure.overlappedv(smua.nvbuffer1)`

smuX.measure.rangeY	X = SMU チャンネル（a または b） Y = SMU 測定関数（v または i）：v = 電圧、i = 電流
属性値	電圧または電流測定に使用する固定測定レンジ。
使用法	`rangeval = smuX.measure.rangeY` -- 測定レンジ読み込み `smuX.measure.rangeY = rangeval` -- 測定レンジ書き込み `rangeval` 測定で予想される電圧 / 電流レンジをセットします。
備考	• この属性値を読み取ることにより、SMU が現在使用している測定レンジの正のフルスケール値を得ることができます。 • この属性値に値をセットすると、その値が収まる充分な余裕を持つ固定レンジが SMU に設定されます。すなわち、rangeval で指定された値を測定する最適レンジを装置が選択してくれます。 • この属性値の主たる目的は測定装置自体が測定レンジを選択するために消費する時間を無くすことにあります。ただし、固定レンジを選択することによりレンジの自動切換えが起こりませんから、オーバーレンジ条件が発生する可能性があります（たとえば、6V レンジで 10.0V を測定するとオーバーレンジが発生します）。オーバーレンジが発生したときは値 9.91000E+37 が返されます。 • ソース機能と測定機能が同じ種類（例えば電圧ソースで電圧測定）である場合、測定レンジは同じソースレンジにロックされます。しかし、その状態でソース機能を電流に切換えたとしても電圧測定レンジはその値を保ち、現在の電圧測定レンジがそのまま継続して使用されます。たとえば、ソース機能として電圧が選択されており、ソースレンジが 1V であるときに測定レンジが 6V であるとします。このときは、ソースレンジが 1V であることから、SMU は 1V レンジを使用して電圧を測定します。ここで機能を電流に切換えると、測定は 6V レンジで実行されます。 • ある機能に対して明示的にソースまたは測定レンジを設定すると、その機能の自動レンジ切換えは解除されます。自動レンジ切換え機能は個々のソース、測定機能（電圧ソース、電流ソース、電圧測定、電流測定）ごとに独立してコントロールされます。デフォルト設定ではすべての機能の自動レンジ切換えがアクティブになっています。 • 出力 OFF の状態でレンジを切換えたとしてもハードウェアの設定はその時点では更新されません。ただし、クエリコマンドに対しては出力が ON に変化したときに使用される予定のレンジ設定を返します。出力が ON の状態でレンジを設定した場合には新しい値が直ちに有効になります。 • ソース側の自動レンジ切換えがアクティブである場合は、出力レベルがレンジをコントロールします。レベル設定後にレンジの問合せを行うと、装置がそのソースレベルに適当と判断したレンジ値が返されます。 • 測定側の自動レンジ切換えがオンになっていても、実際に切換えが起るのは測定の取り込みが行われるときです。したがって、測定後にレンジを問い合わせることによってその測定に実際に使われたレンジを取得することができます。
詳細	セクション 6「レンジ」参照。
参照	smuX.measure.autorangeY
例	SMU A 用に 1V 測定レンジを選択します。 `smua.measure.rangev = 0.5`

smuX.measure.rel.enableY	X = SMU チャンネル（a または b） Y = SMU 測定関数（v、i、r 、p）： v = 電圧、i = 電流、r = 抵抗、p = 電力
属性値	相対測定のコントロール（on/off）。
使用法	`rel = smuX.measure.rel.enableY`　-- 相対測定の状態読み取り `smuX.measure.rel.enableY = rel`　-- 相対測定の状態書き込み rel に次のいずれかの値をセットします。 0 または smuX.REL_OFF　相対測定を OFF にします。 1 または smuX.REL_ON　相対測定を ON にします。
備考	相対測定を ON にすると、それ以後取り込まれるすべての読み取り値には特定の相対オフセット値（smuX.measure.rel.levelY 参照）を使用したオフセット処理が施されます。具体的には、相対測定では次式による計算が施された結果が返されます。 相対読み取り値 = 実際に測定した読み取り値 － 相対オフセット値
詳細	セクション 6「リラティブ」参照。
参照	smuX.measure.rel.levelY
例	SMU A の相対電圧測定を ON にします。 `smua.measure.rel.enablev = smua.REL_ON`

smuX.measure.rel.levelY	X = SMU チャンネル（a または b） Y = SMU 測定関数（v、i、r 、p）： v = 電圧、i = 電流、r = 抵抗、p = 電力
属性値	相対測定で使用するオフセット値。
使用法	`relval = smuX.measure.rel.levelY`　-- 相対オフセットレベル読み取り `smuX.measure.rel.levelY = relval`　-- 相対オフセットレベル書き込み `relval`　相対オフセット値
備考	相対測定を ON にすると（smuX.measure.rel.enableY 参照）、それ以後取り込まれるすべての読み取り値には特定の相対オフセット値を使用したオフセット処理が施されます。具体的には、相対測定では次式による計算が施された結果が返されます。 相対読み取り値 = 実際に測定した読み取り値 － 相対オフセット値
詳細	セクション 6「リラティブ」参照。
参照	smuX.measure.rel.enableY
例	電圧測定を行い、その結果を相対オフセット値として使用します。 `smua.measure.rel.levelv = smua.measure.v()`

smuX.measure.Y **smuX.measure.iv**	X = SMU チャンネル（a または b） Y = SMU 測定関数（v、i、r、p）： v = 電圧、i = 電流、r = 抵抗、p = 電力
機能	少なくとも 1 回の測定を行います。

使用法	この関数は次の 3 通りの方法で使用します。
	`reading = smuX.measure.Y()`
	`reading = smuX.measure.Y(rbuffer)`
	`reading = smuX.measure.iv(ibuffer, vbuffer)`
	`reading` 測定プロセスで得られた最も新しい読み取り値を返します。
	`rbuffer` 全ての読み取り値を保存する読み取りバッファオブジェクト
	`ibuffer` 電流読み取り値を保存する読み取りバッファオブジェクト
	`vbuffer` 電圧読み取り値を保存する読み取りバッファオブジェクト

備考	• この関数は読み取り値の中の最も新しい値だけを reading に返します。測定で得られたそれ以外の情報を使用したい場合は読み込みバッファを参照します。装置が複数の読み取り値を返すように設定された状態で測定要求が発生すると、rbuffer がすでに存在する場合はそのバッファにすべての読み取り値が書き込まれ、最後に行われた測定値だけが reading に返されます。 • smuX.measure.iv 関数は電流と電圧両方の読み取り値をそれぞれのバッファに保存します（電流値、続いて電圧値が別々なバッファに保存されます）。 • 何回の測定を実行するのかを決定するのは smuX.measure.count 属性値です。バッファを使用している場合は、バッファに保存する読み取り値の数もこの属性値によって決定されます。
詳細	セクション 7「読み込みバッファ」参照。
参照	smuX.nvbufferY, smuX.nvbufferY

例	SMU A を使用して 10 回の電圧測定を行い、その結果をバッファに保存します。
	`smua.measure.count = 10`
	`smua.measure.v(smua.nvbuffer1)`

smuX.measureYandstep **smuX.measureivandstep**	X = SMU チャンネル（a または b） Y = SMU 測定関数（v、i、r、p）： v = 電圧、i = 電流、r = 抵抗、p = 電力
機能	1 回または 2 回の測定を行い、続いてソースをステップ変化させます。

使用法	この関数の使用法には次の 2 通りがあります。
	`reading = smuX.measureYandstep(sourcevalue)`
	`readings = smuX.measureivandstep(sourcevalue)`
	`reading` ソースをステップ変化させる前に測定読み取り値を返します。
	`readings` ソースをステップ変化させる前に 2 つの測定読み取り値を返します。
	`sourcevalue` 測定後、ソースに設定する値

備考	• smuX.measureYandstep 関数は 1 回の測定を行い、その後ソースに sourcevalue をセットします。smuX.measureivandstep 関数もこれに良く似ていますが、電流（i）と電圧（v）各 1 回、都合 2 回の測定を行う点が異なります。 • 指定するソース値は選択されているソース機能に適合したものでなければなりません。たとえば、電圧ソース機能が選択されているならば、sourcevalue は新しい電圧レベルと解釈されます。 • この関数の使用前にソース側と測定側の自動レンジ機能を解除しておく必要があります。 • この関数はソースメジャーの高速ループ実行を目的として提供されている機能であり、ソースをステップ変化させる直前に測定を行います。この関数の実行前、またはこの関数を含むループ処理の実行前に必ずソースを初期値に設定しておく必要があります。
参照	smuX.measure.Y

例	この例では、ソース値 0V を初期値として、測定とステップ変化を繰り返しながら電流測定を行います。
	1 回の電流測定が終わるごとにソースを 100mV だけステップ変化させて次回の電流測定へ移行します。ソースが最終レベル（1V）に達した段階で最後の電流測定を行います。
	`local ivalues = {}`
	
	`smua.source.rangev = 1`
	`smua.source.levelv = 0`
	`smua.measure.rangei = 0.01`
	`smua.source.output = smua.OUTPUT_ON`
	`for index = 1, 10 do`
	`  ivalues[index] = smua.measureiandstep(index / 10)`
	`end`
	`ivalues[11] = smua.measure.i()`

smuX.nvbufferY	X = SMU チャンネル（a または b） Y = NV バッファ（1 または 2）
属性値	不揮発性読み込みバッファ。
使用法	`smuX.nvbufferY`
備考	• 読み込みバッファには smuX.nvbuffer1 と smuX.nvbuffer2 の 2 種類が用意されています。 • 測定値を返すすべてのルーチンはこれらの値を読み込みバッファに返すことができます。オーバーラップ測定の場合は常に測定値が読み込みバッファに書き込まれて返されます。同期測定の場合は 1 点だけの測定を直接返すか、もしくはその値を測定コマンドに引き渡すことにより読み込みバッファに書き込みます。 • 不揮発性読み込みバッファは途中で装置電源の入り切りがあったとしても値を保持します。
詳細	p.12-6 の「読み込みバッファ」およびセクション 7 の「読み込みバッファ」を参照。
参照	smuX.makebuffer, smuX.measure.overlappedY, smuX.measure.Y
例	SMU A からの電流読み取り値をバッファ 1 に保存します。 `smua.measure.overlappedi(smua.nvbuffer1)`

smuX.nvbufferY.appendmode	X = SMU チャンネル（a または b） Y = NV バッファ（1 または 2）
属性値	読込みバッファの追加書き込み（アペンド）モード。
使用法	`state = smuX.nvbufferY.appendmode` -- アペンドモード読み込み `smuX.nvbufferY.appendmode = state` -- アペンドモード書き込み state に次のいずれかの値をセットします。 `0` アペンドモード OFF - 新しい測定値データを現在のバッファの内容に上書きします。 `1` アペンドモード ON- 新しい測定データを現在のバッファの内容に追加して書き込みます。
備考	• この属性値に値を割り付けることでバッファのアペンドモードの ON/OFF を切換えます。 • アペンドモードを ON にすると、現在バッファに保存されている読み取り値の数が n であったとすれば、新規測定の最初の値が rb[n+1] に書き込まれます。 • バッファのアペンドモードのデフォルト設定は OFF です。
詳細	p.12-6 の「読み込みバッファ」およびセクション 7 の「読み込みバッファ」を参照。
参照	smuX.measure.overlappedY, smuX.measure.Y, smuX.nvbufferY
例	SMU A からの読み込み値をバッファ 1 の内容に追加書き込みします。 `smua.nvbuffer1.appendmode = 1`

smuX.nvbufferY.basetimestamp	X = SMU チャンネル（a または b） Y = NV バッファ（1 または 2）
属性値	最初の読み取り値を保存した時刻を示すタイムスタンプ（時間の起点は装置への電源投入時）。
使用法	`basetime = smuX.nvbufferY.basetimestamp`
備考	• この属性値を読み取ることにより、バッファに保存された最初の読み取り値（rb[1]）のタイムスタンプ（秒）を取得することができます。このタイムスタンプは、測定実行のために装置に電源を入れてから測定値が保存されるまでの経過秒数を示します。 • この属性値は読み出し専用です。
詳細	p.12-6 の「読み込みバッファ」およびセクション 7 の「読み込みバッファ」を参照。
参照	smuX.measure.overlappedY, smuX.measure.Y, smuX.nvbufferY

例	SMU A のバッファ 1 に保存された最初の読み取り値のタイムスタンプを取得する。
	`basetime = smua.nvbuffer1.basetimestamp`
	`print(basetime)`
	Output: `2.369900+03`
	上の出力例は、タイムスタンプが 2369.9 秒であることを示しています。

smuX.nvbufferY.capacity	X = SMU チャンネル（a または b） Y = NV バッファ（1 または 2）
属性値	バッファ容量。
使用法	`capacity = smuX.nvbufferY.capacity`
備考	• この属性値を読み取ると、バッファに保存可能な読み取り値の数が返されます。 • 基本項目のみを保存するように設定した場合、バッファには 100,000 を超える読み取り値を保存することができます。読み取り値が順次書き込まれていっても全体容量は変化しません。追加項目（タイムスタンプやソース値など）も保存するように設定するとバッファに保存可能なデータ数が減少します。 • この属性値は読み出し専用です。
詳細	p.12-6 の「読み込みバッファ」およびセクション 7 の「読み込みバッファ」を参照。
参照	smuX.measure.overlappedY, smuX.measure.Y, smuX.nvbufferY

例	SMU A のバッファ 1 の容量を読み取ります。
	`capacity = smua.nvbuffer1.capacity`
	`print(capacity)`
	Output: `1.123410+05`

smuX.nvbufferY.clear	X = SMU チャンネル（a または b） Y = NV バッファ（1 または 2）
機能	バッファをクリアします。
使用法	`smuX.nvbufferY.clear()`
備考	この関数は指定されたバッファからすべての読み取り値を消去します。
詳細	p.12-6 の「読み込みバッファ」およびセクション 7 の「読み込みバッファ」を参照。
参照	smuX.measure.overlappedY, smuX.measure.Y, smuX.nvbufferY
例	SMU A のバッファ 1 をクリアします。 `smua.nvbuffer1.clear()`

smuX.nvbufferY.collectsourcevalues	X = SMU チャンネル（a または b） Y=NVバッファ（1または2）
属性値	バッファへのソース値保存の有無。
使用法	`state = smuX.nvbufferY.collectsourcevalues`-- 保存の有無を読み取り `smuX.nvbufferY.collectsourcevalues = state`-- 保存の有無を書き込み state に次のいずれかの値をセットします。 `0` ソース値保存 OFF `1` ソース値保存 ON
備考	• この属性値に書き込む値によってソース値保存の有無を指定します。この属性値を読み取ることにより、ソース値が現在保存されているか否かを知ることができます。 • このモードを ON にすると、ソース値が読み取り値といっしょにバッファに保存されます。このため、1 回の読み取りあたり 4 バイトが余分に必要となります。 • このパラメータは ON または OFF（デフォルト）のいずれかの値をとり、変更できるのはバッファが空のときだけです。 バッファを空にするには smuX.nvbufferY.clear 関数を使用してください。
詳細	p.12-6 の「読み込みバッファ」およびセクション 7 の「読み込みバッファ」を参照。
参照	smuX.measure.overlappedY, smuX.measure.Y, smuX.nvbufferY
例	読み取り値と共にソース値を SMU A バッファ 1 に保存します。 `smua.nvbuffer1.collectsourcevalues = 1`

smuX.nvbufferY.collecttimestamps	X = SMU チャンネル（a または b） Y=NVバッファ（1または2）
属性値	バッファへのタイムスタンプ保存の有無。
使用法	`state = smuX.nvbufferY.collecttimestamps` -- 保存の有無を読み取り `smuX.nvbufferY.collecttimestamps = state` -- 保存の有無を書き込み state に次のいずれかの値をセットします。 `0` タイムスタンプを保存しません（off） `1` タイムスタンプを保存します（on）

備考	• この属性値に書き込む値によってタイムスタンプ保存の有無を指定します。この属性値を読み取ることにり、タイムスタンプが現在保存されているか否かを知ることができます。 • この値を ON にすると、タイムスタンプが読み取り値といっしょにバッファに保存されます。このため、1 回の読み取りあたり 4 バイトが余分に必要となります。最初の読み取り値のタイムスタンプが 0 秒となり、それ以後の読み取り値には保存開始時刻を起点とするタイムスタンプが書き込まれます。 • このパラメータは ON または OFF（デフォルト）のいずれかの値をとり、変更できるのはバッファが空のときだけです。 バッファを空にするには smuX.nvbufferY.clear 関数を使用してください。
詳細	p.12-6 の「読み込みバッファ」およびセクション 7 の「読み込みバッファ」を参照。
参照	smuX.measure.overlappedY, smuX.measure.Y, smuX.nvbufferY
例	読み取り値と共にタイムスタンプを SMU A バッファ 1 に保存します。 `smua.nvbuffer1.collecttimestamps = 1`

smuX.nvbufferY.n	X = SMU チャンネル（a または b） Y = NV バッファ（1 または 2）
属性値	バッファ内に存在する読み取り値の数。
使用法	`bufferreadings = smuX.nvbufferY.n`
備考	• この属性値を読み取ると、現在バッファに書き込まれている読み取り値の数が返されます。 • この属性値は読み出し専用です。
詳細	p.12-6 の「読み込みバッファ」およびセクション 7 の「読み込みバッファ」を参照。
参照	smuX.measure.overlappedY, smuX.measure.Y, smuX.nvbufferY

例	SMU A のバッファ 1 に保存されている読み取り値の数を取得します。
	`bufferreadings = smua.nvbuffer1.n`
	`print(bufferreadings)`
	Output: `1.250000+02`
	上記の出力例は、バッファ内に 125 個の読み取り値が保存されていることを示します。

smuX.nvbufferY.timestampresolution	X = SMU チャンネル（a または b） Y = NV バッファ（1 または 2）
属性値	タイムスタンプ分解能。
使用法	`tsres = smuX.nvbufferY.timestampresolution` -- タイムスタンプ保存の状態読み取り `smuX.nvbufferY.timestampresolution = tsres` -- タイムスタンプ保存の状態書き込み `tsres` タイムスタンプ分解能（秒）
備考	• この属性値に値を書き込むことによりタイムスタンプの分解能が決まります。また、この属性値を読み取ることによりタイムスタンプ分解能を取得することができます。 • タイムスタンプ分解能のデフォルト値は 0.000001 秒（1 μs）です。デフォルト分解能を使用すると、バッファに最高 71 分までの重複しないタイムスタンプを保存することができます。長時間を要する試験の場合は分解能を粗くしてください。 • 設定可能な最小値は 1 μs です。 • 設定された値を 2 μs の偶数累乗に丸めた値が使用されます。
詳細	p.12-6 の「読み込みバッファ」およびセクション 7 の「読み込みバッファ」を参照。
参照	smuX.measure.overlappedY, smuX.measure.Y, smuX.nvbufferY
例	SMU A バッファのタイムスタンプ分解能を 10 μs に設定します。 `smua.nvbuffer1.timestampresolution = 0.00001`

smuX.reset	X = SMU チャンネル（a または b）
機能	出力をオフにして SMU をデフォルト設定の状態に戻します。
使用法	`smuX.reset()`
備考	SMU をデフォルト設定（表 1-3 参照）の状態に戻します。
詳細	セクション 1 の「デフォルト設定」参照。
参照	reset

smuX.sense	X = SMU チャンネル（a または b）
属性値	センスモード（リモート / ローカル）の状態。
使用法	`sense = smuX.sense` -- センスモード読み込み `smuX.sense = sense` -- センスモード書き込み sense に次のいずれかの値をセットします。 0 または smuX.SENSE_LOCAL ローカルセンシング（2 線式）を選択 1 または smuX.SENSE_REMOTE リモートセンシング（4 線式）を選択 3 または smuX.SENSE_CALA 校正センスモードを選択

備考	• ソースメジャー操作は 2 線式のローカルセンシング方式または 4 線式のリモートセンシング方式のいずれかの接続を用いて行われます。この属性値に書き込むことによってどちらかのモードを選択します。 • smuX.SENSE_CALA は校正のみに使用するモードですから、校正が許可された状態でのみ選択してください。 • ローカルとリモートモード間のセンスモード切換えは出力が ON の状態でも可能です。 • 校正センスモードは出力が ON である間は選択できません。 • reset 関数を実行するとローカルセンスモードが選択されます。
詳細	セクション 3「センシング方法」参照。
例	SMU A のセンシングモードとしてリモートを選択します。 `smua.sense = smua.SENSE_REMOTE`

smuX.source.autorangeY	X = SMU チャンネル（a または b） Y = SMU 測定関数（v または i）: v = 電圧、i = 電流
属性値	ソースの自動レンジコントロール（ON/OFF）。
使用法	`sautorange = smuX.source.autorangeY` -- ソースの自動レンジ設定読み込み `smuX.source.autorangeY = sautorange` -- ソースの自動レンジ設定書き込み autorange に次のいずれかの値をセットします。 0 または smuX.AUTORANGE_OFF　ソース自動レンジ禁止 1 または smuX.AUTORANGE_ON　ソース自動レンジ許可
備考	• この属性値はソース自動レンジの設定状態を示します。SMU ソース回路が固定レンジに設定されていると smuX.AUTORANGE_OFF、自動レンジモードに入っている場合は smuX.AUTORANGE_ON という値を示します。 • この属性値を smuX.AUTORANGE_OFF にセットすると SMU は固定レンジモードに入り、その時点で SMU ソース回路が使用していたレンジがその固定レンジとなります。 • この属性値を smuX.AUTORANGE_ON にセットすることにより SMU のソース回路が自動レンジモードに入ります。ソース出力が ON になっているときは、もし現在の出力が SMU レンジと異なっていれば、直ちに現在の出力レベルに即して最も適切なレンジに自動的に切り換わります。 • ソースレベルを前面パネルから変更すると自動レンジが解除されます。
詳細	セクション 6 の「レンジ」参照。
参照	smuX.measure.autorangeY, smuX.source.rangeY
例	SMU A の電圧ソースを自動レンジで動作させます。 `smua.source.autorangev = smua.AUTORANGE_ON`

smuX.source.calibrateY	X = SMU チャンネル（a または b） Y = SMU 測定関数（v または i）： v = 電圧、i = 電流
機能	新しいソース校正定数を発生させ、それをアクティブにします。
使用法	`smuX.source.calibrateY(range, cp1expected, cp1reference, cp2expected, cp2reference)` `range` 校正の対象となる測定レンジ `cp1expected` 校正ポイント 1 としてプログラムされたソース値 `cp1reference` 校正ポイント 1 を外部で測定した基準測定値 `cp2expected` 校正ポイント 2 としてプログラムされたソース値 `cp2reference` 校正ポイント 2 を外部で測定した基準測定値
備考	• この関数は与えられたレンジで使用する校正定数を新たに作成して、それらの値をアクティブにします。 ソースは正極性と負極性で別々に校正を行わなければなりません。あるレンジを正極性で校正する場合は正の値を用い、負極正で校正するには負の値を使用します。 • 標準的には 2 つの校正ポイントを使用し、校正ポイント 1 をゼロ近傍、校正ポイント 2 をフルスケールの 90% 近傍に設定します。SMU のソースが正しい校正状態から大幅に離れていると思われるときは、フルスケール値を校正ポイント 2 として使用しないでください。 • 0.0 は正の値と解釈されますから、この値を負極性校正の校正ポイントとして使用しないでください。 • この 2 点の基準値は、現在アクティブな校正セットをソースに共通に適用して測定する必要があります。たとえば、ある値を出力して測定を行い、アクティブな校正セットを変更せずにこのコマンドを発行してください。 • 新しい校正定数は作成後直ちに有効となりますが、そのまま自動的に不揮発性メモリに保存される訳ではありません。smuX.cal.save を使用して新しい定数を不揮発性メモリへ保存してください。現在アクティブな校正定数は、装置の電源を切って再投入するか、もしくは smuX.cal.restore 関数を使用して不揮発性メモリから別な校正セットを呼び出すまで有効です。 • smuX.cal.unlock が成功裏に呼び出されるまでこの関数は機能しません。
詳細	セクション 17（校正）参照。
参照	smuX.cal.restore, smuX.cal.save, smuX.makebuffer, smuX.measure.calibrateY

smuX.source.compliance	X = SMU チャンネル（a または b）
属性値	ソースのコンプライアンス状態。
使用法	`compliance = smuX.source.compliance`
備考	• この属性値を使用してソースのコンプライアンス状態を読み取ります。リミット機能がソースをコントロールしていれば true（ソースはコンプライアンス状態にある）、ソース機能が出力をコントロールしていれば false（ソースはコンプライアンス状態から外れている）を返します。 • この属性値は読み出し専用です。この属性値に値を書き込もうとするとエラーが発生します。 • この属性値を読み取ることにより、ステータスモデルと前面パネルのコンプライアンス情報が更新されます。
詳細	セクション 4（基本操作）および付録 D（ステータスモデル）参照。
参照	smuX.source.limitY

例	SMU A ソースのコンプライアンス状態を読み取ります。
	`compliance = smua.source.compliance`
	`print(compliance)`
	`Output: true`
	上記の出力例は、ソースのレベルが電圧リミット（電流ソースとして設定されている場合）に到達したか、または電流リミット（電圧ソースとして設定されている場合）に到達したことを示しています。

smuX.source.func	X = SMU チャンネル（a または b）
属性値	ソース機能。
使用法	`iv = smuX.source.func` -- ソース機能読み取り `smuX.source.func = iv` -- ソース機能書き込み iv に次のいずれかの値をセットします。 0 または smuX.OUTPUT_DCAMPS 電流ソース機能を選択します 1 または smuX.OUTPUT_DCVOLTS 電圧ソース機能を選択します
備考	• この属性値を読み取ることにより現在ソースが設定されている出力機能を取得します。また、この属性値に書き込むことで SMU を電圧ソースまたは電流ソースに設定します。 • reset 関数は電圧機能を選択します。
詳細	セクション 4（基本操作）参照。
参照	smuX.source.output, smuX.source.levelY
例	SMU A に電流ソース機能を設定します。 `smua.source.func = smua.OUTPUT_DCAMPS`

smuX.source.levelY	X = SMU チャンネル（a または b） Y = SMU 測定関数（v または i）： v = 電圧、i = 電流
属性値	ソースレベル。
使用法	`sourceval = smuX.source.levelY` -- ソース値読み取り `smuX.source.levelY = sourceval` -- ソース値書き込み `sourceval` 電圧を 0 ～± 40（V）の範囲で設定します。 電流を 0 ～± 3（A）の範囲で設定します。
備考	• この属性値は電圧 / 電流ソースのソースレベルを規定します。 • ソースが電圧源として設定されており、かつ出力が ON になっている場合は、smuX.source.levelv で新しく指定したレベルが即時出力されます。出力が OFF の場合、もしくはソースが電流源として設定されていた場合は、ソースが電圧源に設定変更されて出力が ON になった時点で指定の電圧レベルが出力されます。 • ソースが電流源として設定されており、かつ出力が ON になっている場合は、smuX.source.leveli で新しく指定したレベルが即時出力されます。出力が OFF の場合、もしくはソースが電圧源として設定されていた場合は、ソースが電流源に設定変更されて出力が ON になった時点で指定の電流レベルが出力されます。 • sourceval の符号はソースの極性を表します。正の値を指定すると、Low 端子に対して High 端子が相対的に高い電圧 / 電流を発生させ、負の値を指定するとその逆に High 端子が Low 端子よりも相対的に低い値を出力します。 • reset 関数はソースレベルを 0V、0A に設定します。
詳細	セクション 4（基本操作）参照。
参照	smuX.source.func, smuX.source.output
例	SMU A の V ソースを 1V に設定します。 `smua.source.levelv = 1`

smuX.source.limitY	X = SMU チャンネル（a または b） Y = SMU 測定関数（v または i）： v = 電圧、i = 電流
属性値	コンプライアンス リミット。
使用法	`limit = smuX.source.limitY` -- コンプライアンスリミット読み取り `smuX.source.limitY = limit` -- コンプライアンスリミット書き込み `limit` 電圧コンプライアンス：0 ～± 40（V） 電流コンプライアンス：0 ～± 3（A）
備考	• 電圧ソースの電流出力リミットを設定するには属性値 smuX.source.limiti を用い、電流ソースの電圧出力リミット設定には smuX.source.limitv を使用します。SMU は常にそのリミット設定に対応したソースレンジを選択します（自動レンジ設定）。 • この属性値は、実際にソースを ON にする前に、試験シーケンスを用いて設定してください。 • この属性値を読み取ることにより現在設定されているコンプライアンス値を取得することができます。ソースコンプライアンスの状態取得には smuX.source.compliance を使用します。
詳細	セクション 4（基本操作）参照。

参照	smuX.source.compliance, smuX.source.func, smuX.source.output
例	SMU A の V コンプライアンスを 30V に設定します。 `smua.source.limitv = 30`

smuX.source.lowrangeY	X = SMU チャンネル（a または b） Y = SMU 測定関数（v または i）： v = 電圧、i = 電流
属性値	自動レンジ機能に選択を許す最低ソースレンジ。
使用法	`rangeval = smuX.source.lowrangeY` -- 最低レンジ読み取り `smuX.source.lowrangeY = rangeval` -- 最低レンジ書き込み `rangeval` ソース出力が使用できる最低電圧 / 電流レンジを設定します。
備考	• この属性値を使用してソースの自動レンジが選択可能な最低レンジを指定します。低いレンジの方が一般に長い整定時間を必要とします。レンジ値として大きな値を設定することにより、小さな値をより短い整定時間で出力可能になることがあります。 • 装置を自動レンジに設定したときに、rangeval で指定する最低レンジよりも低いレンジに入っていた場合は自動的に rangeval が指定する最低レンジに切り換わります。
詳細	セクション 6「レンジ」参照。
参照	smuX.source.autorangeY, smuX.source.rangeY
例	SMU A の最低電圧レンジを 1V に設定します。これにより、装置が電圧源供給で 100mV レンジを使用することはなくなります。 `smua.source.lowrangev = 1`

smuX.source.offmode	X = SMU チャンネル（a または b）
属性値	ソース出力 OFF モード。
使用法	`offmode = smuX.source.offmode` -- 出力 OFF モード読み込み `smuX.source.offmode = offmode` -- 出力 OFF モード書き込み offmode に次のいずれかの値をセットします。 0 または smuX.OUTPUT_NORMAL 出力を OFF にした時に 0V を出力します。 1 または smuX.OUTPUT_ZERO OFF の時に出力をゼロ（電圧または電流）にします。 2 または smuX.OUTPUT_HIGH_Z 出力が OFF になった時に出力リレーをオープンにします。

備考	• この属性値を読み取ることによりソースの出力 OFF モードを取得することができます。また、この属性値に値をセットすることによって SMU の出力 OFF モードを設定します。 • offmode のデフォルト値は smuX.OUTPUT_NORMAL です。このモードでは、SMU は 0V を供給するとともに、コンプライアンスを現在のソースレンジの 10% または 100uA のどちらか小さい方に設定します。ソース機能として電圧が選択されている場合、10% コンプライアンスは本質的にコンプライアンス電流の減少を意味します。一方、ソース機能が電流である場合の 10% コンプライアンスは現在供給している電流を上回る場合と下回る場合の両方があり得ます。 • offmode に smuX.OUTPUT_HIGH_Z をセットすると、SMU は出力が OFF になったときに出力リレーをオープンにします。 • offmode に smuX.OUTPUT_ZERO をセットすると、SMU は OUTPUT_NORMAL モードと同様に 0V を出力します。ソース機能が電圧である場合、電流リミットは変更されません。ソース機能が電流である場合、電流リミットは現在のソースレベル、もしくは現在のソースレンジの 10% のいずれか値が大きい方へ設定されます。
詳細	セクション 3「出力 OFF 状態」参照。
参照	smuX.source.output
例	SMU-A の出力 OFF 状態を設定します。 `smua.source.offmode = smua.OUTPUT_HIGH_Z`

smuX.source.output	X = SMU チャンネル（a または b）
属性値	ソース出力コントロール（ON/OFF）。
使用法	`state = smuX.source.output` -- 出力状態読み取り `smuX.source.output = state` -- 出力状態書き込み state に次のいずれかの値をセットします。 0 または smuX.OUTPUT_OFF ソース出力を OFF にします。 1 または smuX.OUTPUT_ON ソース出力を ON にします。
備考	この属性値を読み取ることによりソースの現在の出力状態を取得します。また、この属性値に値をセットすることによってソースの出力状態（ON/OFF）を設定します。OFF がソースのデフォルト設定です。出力が ON にされると、SMU は smuX.source.func の指定に従って電圧または電流を供給します。
詳細	セクション 4（基本操作）参照。
参照	smuX.source.func, smuX.source.offmode
例	SMU A のソース出力を ON にします。 `smua.source.output = smua.OUTPUT_ON`

smuX.source.outputenableaction X = SMU チャンネル（a または b）	
属性値	ソースのインターロック動作。
使用法	`action = smuX.source.outputenableaction` -- イネーブル動作読み込み `smuX.source.outputenableaction = action` -- イネーブル動作書き込み action に次のいずれかの値をセットします。 0 または smuX.OE_NONE 何もしません。 1 または smuX.OE_OUTPUT_OFF ソース出力を OFF にします。
備考	• この属性値は、インターロックラインがアサート、またはアサートが解除されたときに SMU が実行する動作を規定します。デフォルト設定は smuX.OE_NONE です。 • smuX.OE_NONE にセットしておくと、SMU はインターロックラインが Low へ変化（アサート解除）しても何の動作も行いません。 • smuX.OE_OUTPUT_OFF にセットしておいてインターロックラインのアサートを解除すると、SMU は出力を OFF にします（smuX.source.output = smuX.OUTPUT_OFF コマンドを受信したときと同じ動作）。 • インターロックラインが High に戻ったとしても、SMU は自動的には出力を ON に戻しません。 • 出力が ON のときにこの属性値を smuX.OE_OUTPUT_OFF にセットし、その時インターロックラインがアサートされていなければ、出力が直ちに OFF になります。 • インターロックラインの Low への変化が検出されたとしても、実行中のスクリプトは中断されませんが、実行時エラーを発生させることがあります。
詳細	セクション 10「インターロック」参照。
参照	smuX.source.offmode, smuX.source.output
例	インターロックラインが Low へ変化（アサート解除）したときに出力が OFF になるように SMU を設定し直します。 `smua.source.outputenableaction = smua.OE_OUTPUT_OFF`

smuX.source.rangeY	X = SMU チャンネル（a または b） Y = SMU 測定関数（v または i）： v = 電圧、i = 電流
属性値	ソースレンジ。
使用法	`rangeval = smuX.source.rangeY` -- ソースレンジ読み取り `smuX.source.rangeY = rangeval` -- ソースレンジ書き込み `rangeval` 出力するべき電圧 / 電流値。
備考	• この属性値を読み取ることにより SMU が現在使用しているソースレンジの正のフルスケール値を得ることができます。この属性値に値をセットすると、その値を供給するのに充分な余裕を持つ固定レンジが SMU に設定されます。すなわち、rangeval で指定された値を出力する最適レンジを装置が選択してくれます。 • smuX.source.rangeX の主たる目的はソース装置自体が測定レンジを選択するために消費する時間を無くすことにあります。ただし、固定レンジを選択することによりレンジの自動切換えが起らなくなりますから、オーバーレンジ条件が発生する可能性があります（たとえば、6.0V レンジで 10.0V を出力するとオーバーレンジが発生します）。

詳細	セクション 6「レンジ」参照。
参照	smuX.source.autorangeY
例	SMU A 用に 1V ソースレンジを選択します。 `smua.source.rangev = 1`

timer 関数

このグループの関数を使用してタイマを操作します。タイマは、各種操作に必要な時間の測定などに使用されます。このような操作では、最初に timer.reset 関数を用いてタイマをゼロにリセットしてから操作を始め、その終了時に timer.measure.t を使用して経過時間を取得します。

timer.measure.t	
機能	タイマが最後にリセットされて以後の経過時間を測定します。
使用法	`time = timer.measure.t()` `time` 経過時間を秒で返します（時間分解能 1 μ s）。
備考	• この関数はタイマがリセットされてから以後の経過秒数を返します。 • 返される値の時間分解能はリセット以後経過した時間の長さに依存し、最初は 1 μ s 分解能ですが、経過時間が 2.8 分を超えると分解能が切り下げられます。
参照	timer.reset

例	最初にタイマをリセットし、リセット以後の経過時間を測定します。
	`timer.reset()`
	`...`
	`time = timer.measure.t()`
	`print(time)`
	Output: `1.469077e+01`
	上の出力例は、タイマのリセット（timer.reset）後 14.69077 秒が経過した時点で timer.measure.t が実行されたことを示します。

timer.reset	
機能	タイマを 0 秒にリセットします。
使用法	`timer.reset()`
備考	この関数はタイマを 0 秒から再スタートさせます。
参照	timer.measure.t

例	最初にタイマをリセットし、リセット以後の経過時間を測定します。
	`timer.reset()`
	`...`
	`time = timer.measure.t()`
	`print(time)`
	Output: `1.469077e+01`
	上の出力例は、タイマのリセット（timer.reset）後 14.69077 秒が経過した時点で timer.measure.t が実行されたことを示します。

trigger 関数

このグループに属する関数を使用してトリガをコントロールします。

trigger.clear	
機能	コマンドインターフェイスのトリガイベントディテクタをクリアします。
使用法	`trigger.clear()`
備考	トリガイベントディテクタは最後の trigger.wait 呼び出し以後にイベントが検出されたか否かを記憶しています。この関数はそのトリガのイベントディテクタをクリアしてコマンドインターフェイスのトリガイベント履歴を消去します。
詳細	セクション 4「トリガ」参照。
参照	trigger.wait

trigger.wait	
機能	トリガイベントの発生を待ちます。
使用法	`triggered = trigger.wait(timeout)` `timeout` トリガ待ちを行う最長時間（秒） `triggered` トリガを検出すると true を返します。timeout で指定された時間が経過してもトリガを検出できなかった場合は false を返します。
備考	• この関数は、アクティブなコマンドインターフェイスとして GPIB が選択されている場合は timeout（秒）だけ GPIB GET コマンドまたは *TRG メッセージの到着を待ち、それ以外のコマンドインターフェイスでは timeout（秒）だけ *TRG メッセージを待ちます。これらのトリガイベントの少なくとも 1 つがすでに到着していたことを検出した場合、この関数は即時リターンします。 • この関数を使用してトリガ待ちを行った後は、イベントディテクタが自動的にリセットされて次回に備えた準備が行われます。この動作は検出したイベントの数によらず同様です。
詳細	セクション 4「トリガ」参照。

参照	trigger.clear

例	最長 10 秒間だけトリガ待ちを行います。
	`triggered = trigger.wait(10)`
	`print(triggered)`
	Output: `false`
	上の出力例は、タイムアウト（10 秒間）終了までにトリガを検出しなかったことを示しています。

tsplink 関数と属性値

このグループの関数と属性値を使用して 260x 装置にノード番号を割り付け、TSP-Link システムを初期化します。

tsplink.node	
属性値	TSP-Link のノード番号。
使用法	`mynode = tsplink.node` -- ノード番号読み取り `tsplink.node = mynode` -- ノード番号書き込み `mynode` 1 から 64 までの範囲でノード番号を設定します。
備考	• この属性値は TSP-Link のノード番号を設定し、その値を不揮発性メモリに保存します。 • ノード番号の変更後、システム内のいずれかのノードに対して tsplink.reset が実行されることによって新しい設定が有効になります。 • TSP-Link に接続されている各ノードには相互に異なるノード番号を割り付ける必要があります。
詳細	セクション 9（システムの拡張）参照。
参照	tsplink.reset, tsplink.state
例	TSP-Link ノードに番号 2 を割り付けます。 `tsplink.node = 2`

tsplink.reset	
機能	TSP-Link システム内のすべてのノード（装置）を初期化（リセット）します。
使用法	`tsplink.reset()`
備考	この関数は TSP-Link に接続されている他のノードのすべての情報を一旦消去してシステムを再構成します。リモートノードへのアクセスを可能にするには少なくとも 1 回はこの関数を呼び出す必要があります。いずれかの装置のノード番号を変更したときは TSP-Link を再度初期化しなければなりません。
詳細	セクション 9（システムの拡張）参照。
参照	tsplink.node, tsplink.state

tsplink.state	
属性値	TSP-Link のオンライン状態。
使用法	`state = tsplink.state`
備考	• この属性値には TSP-Link のステータス（オンライン / オフライン）が保存されています。このステータスは電源投入直後は “offline” になっており、tsplink.reset が成功裏に実行されると “online” に変化します。 • この属性値は読み出し専用です。
詳細	セクション 9（システムの拡張）参照。
参照	tsplink.node, tsplink.reset

例	TSP-Link のオンライン状態を読み取ります。
	`state = tsplink.state`
	`print(state)`
	Output: `online`

userstring 関数

このグループの関数を使用してユーザ定義文字列の不揮発性メモリへの書き込み、不揮発性メモリからの読み出しを行います。

userstring.add	
機能	ユーザ定義文字列を不揮発性メモリへ追加します。
使用法	`userstring.add(name, value)` `name` 文字列の名前 `value` name に関連付ける文字列
備考	この関数は文字列とその名前を関連付け、ペアとして不揮発性メモリに保存します。その名前を指定して文字列を取り出すには userstring.get 関数を使用します。
参照	userstring.catalog, userstring.delete, userstring.get

例	ユーザ定義文字列を不揮発性メモリに保存します。
	`userstring.add("assetnumber", "236")`
	`userstring.add("department", "Widgets")`
	`userstring.add("contact", "John Doe")`

userstring.catalog	
機能	ユーザ文字列カタログを操作するイテレーターを作成します。
使用法	`for name in userstring.catalog() do ... end`
備考	ユーザ文字列名のカタログにアクセスすることにより、不揮発性メモリに保存されている文字列名の出力 / 削除を可能にします。これらの項目には整列化処理が行われていません。
参照	userstring.add, userstring.delete, userstring.get

例	すべてのユーザ定義文字列を不揮発性メモリから削除します。
	`for name in userstring.catalog() do`
	`  userstring.delete(name)`
	`end`
	不揮発性メモリに保存されているユーザ文字列と名前のペアをすべて出力します。
	`for name in userstring.catalog() do`
	`  print(name .. " = " .. userstring.get(name))`
	`end`
	Output: `department = Widgets`
	`  assetnumber = 236`
	`  contact = John Doe`
	上記の出力例は、userstring.add 関数の「例」で使用したユーザ文字列を一覧表示しています。表示順が追加した順番と一致していないことに注意してください。

userstring.delete	
機能	ユーザ定義文字列を不揮発性メモリから削除します。
使用法	`userstring.delete(name)` `  name` ユーザ文字列の名前
備考	この関数は、指定した名前に関連付けられたユーザ文字列を不揮発性メモリから削除します。
参照	userstring.add, userstring.catalog, userstring.get

例	ユーザ定義文字列を不揮発性メモリから削除します。
	`userstring.delete("assetnumber")`
	`userstring.delete("department")`
	`userstring.delete("contact")`

userstring.get	
機能	ユーザ定義文字列を不揮発性メモリから取り出します。
使用法	`value = userstring.get(name)` `name` ユーザ文字列の名前 `value` name に関連付けられた文字列を返します。
備考	この関数は、指定した名前に関連付けられたユーザ文字列を不揮発性メモリから取り出します。
参照	userstring.add, userstring.catalog, userstring.delete

例	ユーザ文字列を不揮発性メモリから取り出します。
	`value = userstring.get("assetnumber")`
	`print(value)`
	Output: `236`

waitcomplete 関数

この関数は、オーバーラップしたコマンドの実行がすべて完了するのを待ちます。

waitcomplete	
機能	オーバーラップしたコマンドの実行がすべて完了するのを待ちます。
使用法	`waitcomplete()`
備考	この関数は、すでに開始済みで実行がオーバーラップしているコマンドがすべて完了するのを待ちます。

13
ファクトリ スクリプト

セクション 13 のトピックス

はじめに , 13-2
ファクトリ スクリプト , 13-2
KIGeneral, 13-2
フラッシュファームウェアのアップグレード , 13-13

はじめに

260x 型は少なくとも 1 つのファクトリスクリプトがフラッシュ ファームウェア メモリに実装された状態で出荷されます。ファクトリスクリプトは数多くの関数から構成されており、これらの関数は前面パネルの LOAD TEST メニューやリモートプログラミングを使用して呼び出すことができます。

Keithley は逐次追加のファクトリスクリプトを作成しており、これらのファクトリスクリプトを 260x 型フラッシュファームウェア アップグレードキットとして Web サイト（www.keithley.com）からお客様に提供しています。お手許の 260x 型のフラッシュファームウェアをアップグレードする方法の詳細が「フラッシュファームウェアのアップグレード」に説明されています。

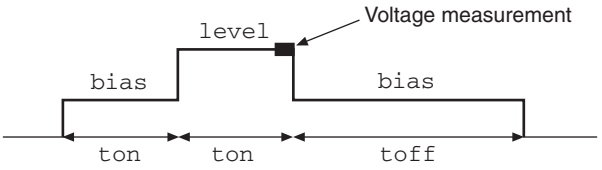
ファクトリ スクリプト

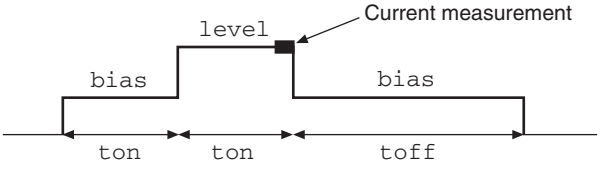
KIGeneral

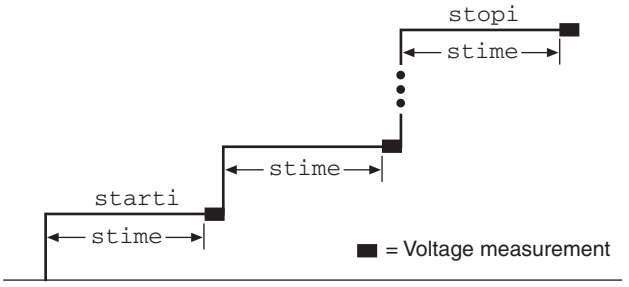
KIGeneral ファクトリスクリプトは以下の関数群によって構成されています。

```
PulseIMeasureV(smu, bias, level, ton, toff, points)
PulseVMeasureI(smu, bias, level, ton, toff, points)
SweepILinMeasureV(smu, starti, stopi, stime, points)
SweepVLinMeasureI(smu, startv, stopv, stime, points)
SweepILogMeasureV(smu, starti, stopi, stime, points)
SweepVLogMeasureI(smu, startv, stopv, stime, points)
SweepIListMeasureV(smu, ilist, stime, points)
SweepVListMeasureI(smu, vlist, stime, points)
```

これらの関数の詳細を以下に続く表で説明します。

TSP プロジェクト名：　KIFactoryGeneral TSP 試験スクリプト名：　KIGeneral Version: 1.0.3RC1	
機能： `PulseIMeasureV(smu, bias, level, ton, toff, points)`	
説明：	指定された回数の電流パルス - 電圧測定サイクルを繰り返します。 •smu が電流値 bias（A）を出力してその状態を ton 秒保持します。 •smu が電流値 level（A）を出力してその状態を ton 秒保持します。 •ソースレベルが level（A）の状態で電圧を測定します。 •smu が電流 bias（A）を出力してその状態を toff 秒保持します。 •上記のパルス - 測定サイクルを points 回繰り返します。 
パラメータ：	`smu, bias, level, ton, toff, points` `smu:`　ソースメータチャンネル（A または B）。関数呼び出し時にパラメータが全部省略されると smua がデフォルトとして選択されます。 `bias:`　バイアルレベル（A） `level:`　パルスレベル（A） `ton:`　パルスの ON 時間（秒） `toff:`　パルスの OFF 時間（秒） `points:`　パルス測定サイクルの回数
データ：	パルス電圧測定値、電流レベル、タイムスタンプが smuX.nvbuffer1 に保存されます。
例：	`PulseIMeasureV(smua, 0.001, 1.0, 20E-3, 40E-3, 10)` SMU A はまず 1 mA を 20ms、続いて 1A を 20ms 出力し、その状態で電圧を測定します。測定後、出力が 1mA に戻して 40ms 保持します。このパルス - 測定プロセスをあと 9 回繰り返します。

TSP プロジェクト名： KIFactoryGeneral TSP 試験スクリプト名： KIGeneral Version: 1.0.3RC1	
機能： `PulseVMeasureI(smu, bias, level, ton, toff, points)`	
説明：	指定された回数の電圧パルス - 電流測定サイクルを繰り返します。 •smu が電圧値 bias（V）を出力して、その状態を ton 秒保持します。 •smu が電圧値 level（V）を出力して、その状態を ton 秒保持します。 •ソースレベルが level（V）の状態で電流を測定します。 •smu が電圧 bias（V）を出力してその状態を toff 秒保持します。 •上記のパルス - 測定サイクルを points 回繰り返します。 
パラメータ：	`smu, bias, level, ton, toff, points` `smu:` ソースメータチャンネル（A または B）。関数呼び出し時にパラメータが全部省略されると smua がデフォルトとして選択されます。 `bias:` バイアルレベル（V） `level:` パルスレベル（V） `ton:` パルスの ON 時間（秒） `toff:` パルスの OFF 時間（秒） `points:` パルス測定サイクルの回数
データ：	パルス電流測定値、電圧レベル、タイムスタンプが smuX.nvbuffer1 に保存されます。
例：	`PulseVMeasureI(smub, -1, 1, 1E-3, 2E-3, 20)` SMU B はまず -1V を 1ms、続いて 1V を 2ms 出力し、その状態で電流を測定します。測定後、出力を -1V に戻して 2ms 保持します。このパルス - 測定プロセスをあと 19 回繰り返します。

TSP プロジェクト名：　KIFactoryGeneral TSP 試験スクリプト名：　KIGeneral Version: 1.0.3RC1	
機能： `SweepILinMeasureV(smu, starti, stopi, stime, points)`	
説明：	線形電流スイープを行い、各ステップ（ポイント）で電圧測定を行います。 •smu に電流値 starti（A）を出力させて、安定化（整定時間 stime 秒）するのを待ってから電圧を測定します。 •smu に次のステップの電流値を出力させて、安定化（整定時間 stime 秒）するのを待ってから電圧を測定します。 •電流ステップ stopi（A）での電圧測定が終わるまで上記のシーケンスを反復します。 線形スイープのステップサイズは次式に従って自動的に計算されます。 ステップ ＝(stopi – starti) / (points – 1) 
パラメータ：	`smu, starti, stopi, stime, points` `smu:` ソースメータチャンネル（A または B）。関数呼び出し時にパラメータが全部省略されると smua がデフォルトとして選択されます。 `starti:` スイープ開始電流（A） `stopi:` スイープ終了電流（A） `stime:` 整定時間（秒）。ソースがステップ変化してから測定実行までの時間 `points:` スイープポイントの数（2 以上）
データ：	電圧測定値、電流ソース値、およびタイムスタンプが smuX.nvbuffer1 に保存されます。
例：	`SweepILinMeasureV(smua, -1E-3, 1E-3, 0, 100)` 100 ポイントの線形電流スイープを実行します（開始電流：-1mA、終了電流：+1mA）。スイープの全ステップ（ポイント）で電圧測定を行います。stime が 0 秒に設定されているため、電流のステップ変化後時間を置かずに電圧測定が行われます。

TSP プロジェクト名： KIFactoryGeneral
TSP 試験スクリプト名： KIGeneral
Version: 1.0.3RC1

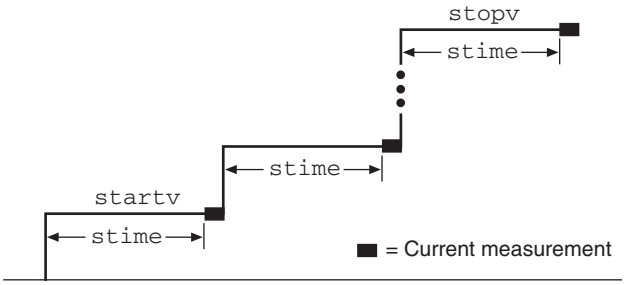
機能：
`SweepVLinMeasureI(smu, startv, stopv, stime, points)`

説明：線形電圧スイープを行い、各ステップ（ポイント）で電流測定を行います。

- smu に電圧値 startv（V）を出力させて、安定化（整定時間 stime 秒）するのを待ってから電流を測定します。
- smu に次のステップの電圧値を出力させて、安定化（整定時間 stime 秒）するのを待ってから電流を測定します。
- 最後の電圧ステップ stopv（V）での電流測定が終わるまで上記のシーケンスを反復します。

線形スイープのステップサイズは次式に従って自動的に計算されます。

ステップ = (stopv – startv) / (points – 1)

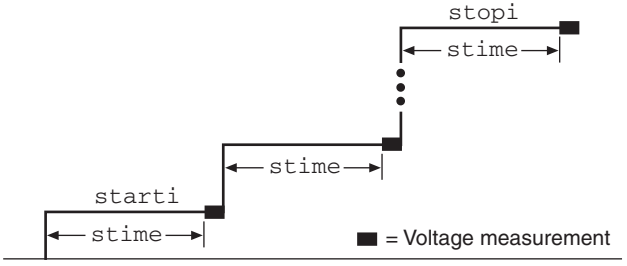


パラメータ：`smu, startv, stopv, stime, points`

- `smu`: ソースメータチャンネル（A または B）。関数呼び出し時にパラメータが全部省略されると smua がデフォルトとして選択されます。
- `starti`: スイープ開始電圧（V）
- `stopi`: スイープ終了電圧（V）
- `stime`: 整定時間（秒）。ソースがステップ変化してから測定実行までの時間。
- `points`: スイープポイントの数（2 以上）

データ：電流測定値、電圧ソース値、およびタイムスタンプが smuX.nvbuffer1 に保存されます。

例：
```
SweepVLinMeasureI(smua, -1, 1, 1E-3, 1000)
```
1000 ポイントの線形電圧スイープを実行します（開始電圧：-1V、終了電圧：+1V）。電流測定は、すべてのスイープステップ（ポイント）で、1ms の整定時間終了後に行われます。

TSP プロジェクト名：　KIFactoryGeneral TSP 試験スクリプト名：　KIGeneral Version: 1.0.3RC1	
機能： `SweepILogMeasureV(smu, starti, stopi, stime, points)`	
説明：	対数電流スイープを行い、各ステップ（ポイント）で電圧測定を行います。 •smu に電流値 starti（A）を出力させて、安定化（整定時間 stime 秒）するのを待ってから電圧を測定します。 •smu に次のステップの電流値を出力させて、安定化（整定時間 stime 秒）するのを待ってから電圧を測定します。 •最終電流ステップ stopi（A）での電圧測定が終わるまで上記のシーケンスを反復します。 各ステップのソースレベル（SourceStepLevel）は次式に従って自動的に計算されます。 **MeasurePoint** = 測定を構成するステップポイント数。たとえば、5 ポイントスイープ（points = 5）ならば MeasurePoint 1、2、…5 で測定が行われます。 **LogStepSize** = (log10(`stopi`) – log10(`starti`)) / (`points` – 1) **LogStep** = (MeasurePoint – 1) x (LogStepSize) **SourceStepLevel** = antilog(LogStep) x `starti` 
パラメータ：	`smu, starti, stopi, stime, points` `smu:`　ソースメータチャンネル（A または B）。関数呼び出し時にパラメータが全部省略されると smua がデフォルトとして選択されます。 `starti:`　スイープ開始電流（A） `stopi:`　スイープ終了電流（A） `stime:`　整定時間（秒）。ソースがステップ変化してから測定実行までの時間。 `points:`　スイープ測定ポイントの数（2 以上）
データ：	電圧測定値、電流ソース値、およびタイムスタンプが smuX.nvbuffer1 に保存されます。

例：

```
SweepILogMeasureV(smua, 0.01, 0.1, 0.001, 5)
```

この関数は 5 ポイントの対数電流スイープを実行します（開始電流：10mA、終了電流：100mA）。スイープの全ステップ（ポイント）で電圧測定を行います。ソースのステップ変化後は 1ms の整定時間を置いてから測定を行います。

対数値とそれに対応するソースレベルを、5 ポイント対数スイープを例として次に示します。

MeasurePoint	LogStepSize	LogStep	SourceStepLevel
1	0.25	0.0	0.01A
2	0.25	0.25	0.017783A
3	0.25	0.5	0.031623A
4	0.25	0.75	0.056234A
5	0.25	1.0	0.1A

TSP プロジェクト名：　KIFactoryGeneral
TSP 試験スクリプト名：　KIGeneral
Version: 1.0.3RC1

機能：
`SweepVLogMeasureI(smu, startv, stopv, stime, points)`

説明：

対数電圧スイープを行い、各ステップ（ポイント）で電流測定を行います。

- smu に電圧値 startv（V）を出力させて、安定化（整定時間 stime 秒）するのを待ってから電流を測定します。
- smu に次のステップの電圧値を出力させて、安定化（整定時間 stime 秒）するのを待ってから電流を測定します。
- 最終電圧ステップ stopv（V）での電流測定が終わるまで上記のシーケンスを反復します。

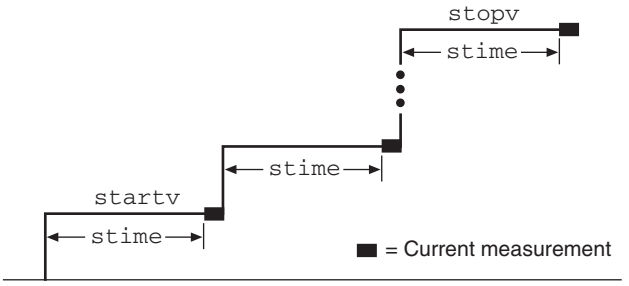
各ステップのソースレベル（SourceStepLevel）は次式に従って自動的に計算されます。

MeasurePoint = 測定を構成するステップポイント数。たとえば、5 ポイントスイープ（points = 5）ならば MeasurePoint 1、2、…5 で測定が行われます。

LogStepSize = (log10(`stopv`) – log10(`startv`)) / (`points` – 1)

LogStep = (MeasurePoint – 1) x (LogStepSize)

SourceStepLevel = antilog(LogStep) x `startv`



パラメータ：

`smu, startv, stopv, stime, points`

`smu:`　ソースメータチャンネル（A または B）。関数呼び出し時にパラメータが全部省略されると smua がデフォルトとして選択されます。

`startv:`　スイープ開始電圧（V）

`stopv:`　スイープ終了電圧（V）

`stime:`　整定時間（秒）。ソースがステップ変化してから測定実行までの時間。

`points:`　スイープ測定ポイントの数（2 以上）

データ：

電流測定値、電圧ソース値、およびタイムスタンプが smuX.nvbuffer1 に保存されます。

例：
```
SweepVLogMeasureI(smua, 1, 10, 0.001, 5)
```
この関数は 5 ポイントの対数電圧スイープを実行します（開始電圧：1V、終了電圧：10V）。スイープの全ステップ（ポイント）で電流測定を行います。ソースのステップ変化後は 1ms の整定時間を置いてから測定を行います。

対数値とそれに対応するソースレベルを、5 ポイント対数スイープを例として次に示します。

MeasurePoint	LogStepSize	LogStep	SourceStepLevel
1	0.25	0.0	1.0000V
2	0.25	0.25	1.7783V
3	0.25	0.5	3.1623V
4	0.25	0.75	5.6234V
5	0.25	1.0	10.000V

TSP プロジェクト名：　KIFactoryGeneral TSP 試験スクリプト名：　KIGeneral Version: 1.0.3RC1	
機能： `SweepIListMeasureV(smu, ilist, stime, points)`	
説明：	リストに従って電流スイープを行い、各ステップ（ポイント）で電圧測定を行います。 •smu にリスト（ilist）の最初の電流値を出力させて、安定化（整定時間 stime 秒）するのを待ってから電圧を測定します。 •smu にリスト（ilist）の次の電流値を出力させて、安定化（整定時間 stime 秒）するのを待ってから電圧を測定します。 •リストに記載の最後の電流ステップでの電圧測定が終わるまで上記のシーケンスを反復します。測定するリストの最後のポイントが points に対応します。

パラメータ：	`smu, ilist, stime, points`
	`smu:` ソースメータチャンネル（A または B）。関数呼び出し時にパラメータが全部省略されると smua がデフォルトとして選択されます。
	`ilist:` 電流ソース値を任意に記述したリスト：
	`ilist = {value1, value2, ....valueN}`
	`stime:` 整定時間（秒）。ソースがある値を出力してから測定実行までの時間。
	`points:` スイープ測定ポイントの数（2 以上）

データ：	電圧測定値、電流ソース値、およびタイムスタンプが smuX.nvbuffer1 に保存されます。

例：	`myilist = {-100E-9, 100E-9, -1E-6, 1E-6, -1E-3, 1E-3}`
	`SweepIListMeasureV(smua, myilist, 500E-6, 6)`
	この関数は myilist に記載された最初の値からスタートして 6 ポイントのリスト電流スイープを行います。スイープの全ステップ（ポイント）で電圧測定を行います。ソースのステップ変化後は 500 μs の整定時間を置いてから測定を行います。

TSP プロジェクト名： KIFactoryGeneral TSP 試験スクリプト名： KIGeneral Version: 1.0.3RC1	
機能： `SweepVListMeasureI(smu, vlist, stime, points)`	
説明：	電圧リストスイープを行い、各ステップ（ポイント）で電流測定を行います。 •smu にリスト（vlist）の最初の電圧値を出力させて、安定化（整定時間 stime 秒）するのを待ってから電流を測定します。 •smu にリスト（vlist）の次の電圧値を出力させて、安定化（整定時間 stime 秒）するのを待ってから電流を測定します。 •リストに記載の最後の電圧ステップでの電流測定が終わるまで上記のシーケンスを反復します。測定するリストの最後のポイントが points に対応します。

パラメータ：	`smu, vlist, stime, points`
	`smu:` ソースメータチャンネル（A または B）。関数呼び出し時にパラメータが全部省略されると smua がデフォルトとして選択されます。
	`vlist:` 電圧ソース値を任意に記述したリスト：
	`vlist = {value1, value2, ....valueN}`
	`stime:` 整定時間（秒）。ソースがある値を出力してから測定実行までの時間。
	`points:` スイープ測定ポイントの数（2 以上）

データ：	電流測定値、電圧ソース値、およびタイムスタンプが smuX.nvbuffer1 に保存されます。

例：	`myvlist = {-0.1, 0.1, -1, 1, -6, 6, -40, 40, 0, 0}`
	`SweepVListMeasureI(smua, myvlist, 500E-6, 10)`
	この関数は myvlist に記載された最初の値からスタートして 10 ポイントのリスト電圧スイープを行います。スイープの全ステップ（ポイント）で電流測定を行います。ソースのステップ変化後は 500 μ s の整定時間を置いてから測定を行います。

フラッシュファームウェアのアップグレード

Keithley は逐次追加のファクトリスクリプトを作成しており、これらのファクトリスクリプトは 260x 型フラッシュファームウェア アップグレードキットとして Web サイト（www.keithley.com）からお客様に提供しています。

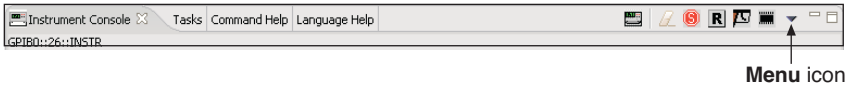
注意 **外部回路が入出力端子に接続された状態でフラッシュアップグレードを行うと装置や DUT に損傷を与える可能性があります。フラッシュアップグレード実行の前に必ず入出力端子の接続を取り外してください。**

最初に新しいフラッシュファイルを Keithley の Web サイトからダウンロードしておき、続いて TSB（Test Script Builder）を以下の順に操作して 260x 型のファームウェアをアップグレードします。

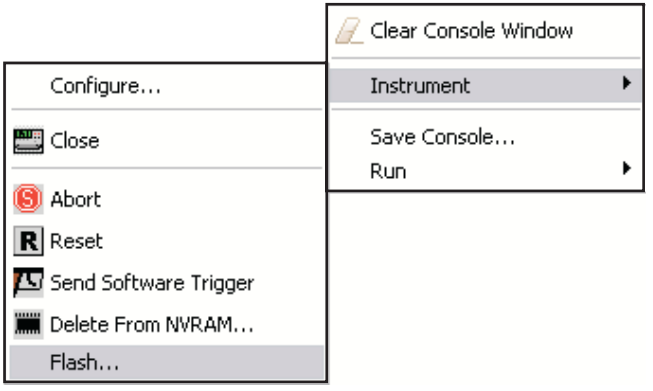
1. PC デスクトップ上の Test Script Builder アイコンをダブルクリックします。
2. Instrument Console ツールバー上の Open Instrument アイコンをクリックし、続いて Select Instrument Resource ダイアログボックスを使用して通信インターフェイスを選択します。通信回路を開く方法の詳細については図 2-6 を参照してください。



3. Instrument Console ツールバーの Menu アイコンをクリックしてメニューを表示させます。



4. 表示されたドロップダウンメニューの中から Instrument を選選択し、続いて Flash をクリックします。



5. Select A Firmware Data File ダイアログのブラウザを使用して新しいファームウェアのファイル名を選択し、Open をクリックして 260x 型のファームウェアをアップグレードします。

14
ディスプレイの操作
（ディスプレイを操作する関数と属性値）

セクション 14 のトピックス

display 関数と属性値 , 14-2
ディスプレイの持つ機能 , 14-3
- ディスプレイ画面 , 14-3
- 測定関数 , 14-3
- 表示分解能 , 14-4

メッセージ表示 , 14-4
- 画面クリア , 14-5
- カーソル位置 , 14-5
- テキストメッセージの表示 , 14-6

入力プロンプト , 14-8
- メニュー , 14-8
- パラメータ入力プロンプト , 14-10

表示ランプ , 14-11
LOCAL ロックアウト , 14-12
テストメニューのローディング , 14-13
- ユーザスクリプトの保存 , 14-13
- USER TESTS メニューの項目追加 , 14-13
- USER TESTS メニューの項目削除 , 14-14
- 前面パネルからテストを実行 , 14-14

ディスプレイトリガ , 14-15
キープレスコード , 14-16
- キーコードを送る , 14-16
- キープレスコードの取り込み , 14-17

display 関数と属性値

このセクションで説明するディスプレイ操作を行うには display 関数とそれに関連した属性値を使用します。display 関数と属性値（アルファベット順）、およびその関数 / 属性値が説明されているセクショントピックスとの対応を表 14-1 にまとめてあります。

display 関数と属性値についてはセクション 12 でも追加情報が説明されています。

表 14-1
関数 / 属性値とセクショントピックス対照表

関数 / 属性値	セクションの項目名
`display.clear`	画面クリア、p.14-5
`display.getannunciators`	表示ランプ、p.14-11
`display.getcursor`	カーソル位置、p.14-5
`display.gettext`	テキストメッセージ表示、p.14-6
`display.input`	キープレスコードの取り込み、p.14-17
`display.inputvalue`	パラメータ入力プロンプト、p.14-10
`display.loadmenu.add` `display.loadmenu.delete`	テストメニューのローディング、p.14-13
`display.locallockout`	LOCAL ロックアウト、p.14-12
`display.menu`	メニュー、p.14-8
`display.prompt`	パラメータ入力プロンプト、p.14-10
`display.screen`	ディスプレイ画面、p.14-3
`display.sendkey`	キーコードを送る、p.14-16
`display.setcursor`	カーソル位置、p.14-5
`display.settext`	テキストメッセージ表示、p.14-6
`display.smuX.digits`	表示分解能、p.14-4
`display.smuX.measure.func`	測定機能、p.14-3
`display.trigger.clear` `display.trigger.wait`	ディスプレイトリガ、p.14-15

ディスプレイの持つ機能

ディスプレイ画面

260x 型のディスプレイはソースメジャー値 / 読み取り値ばかりでなく、ユーザが定義したメッセージも表示します。画面の表示内容について以下のオプションが用意されています。

- ソースメジャー、コンプライアンス画面：
  - ソースとコンプライアンス値、および SMU A 測定読み取り値表示
  - ソースとコンプライアンス値、および SMU B 測定読み取り値表示
- ソースメジャー画面 - ソース値および SMU A と SMU B の測定読み取り値を表示
- ユーザ画面 - ユーザ定義メッセージとプロンプトを表示

表示内容を選択するには属性値 display.screen を使用します。

```
display.screen = displayId
```

ここで、displayId には次の何れかの値（または定数名）を設定します。

0 または display.SMUA
1 または display.SMUB
2 または display.SMUA_SMUB
3 または display.USER

ディスプレイ画面例：

次のコマンドを実行すると、ディスプレイには SMU A のソースメジャー値とコンプライアンス値が表示されます。

```
display.screen = display.SMUA
```

測定関数

ソースメジャー画面を選択しておけば、そこに測定読み取り値を希望の単位（電圧、電流、抵抗、ワット）で表示することができます。

表示する測定機能を選択するには属性値 display.smuX.measure.func を使用します。

```
display.smuX.measure.func = function
```

ここに、smuX は smua または smub のいずれかを表します。
function には次のいずれかの値を設定します。

0 または display.MEASURE_DCAMPS
1 または display.MEASURE_DCVOLTS
2 または display.MEASURE_OHMS
3 または display.MEASURE_WATTS

測定関数の例：

次のコマンドを実行すると、SMU A は抵抗測定値を表示します。

```
display.smua.measure.func = display.MEASURE_OHMS
```

表示分解能

測定値の表示に使用する表示分解能として 4-1/2、5-1/2、または 6-1/2 桁のいずれかを選択します。

測定値の表示分解能を指定するには属性値 display.smuXdigits を使用します。

```
display.smuX.digits = digits
```

ここに、smuX は smua または smub のいずれかを表します。
digits には次のいずれかの値を設定します。
4 または display.DIGITS_4_5
5 または display.DIGITS_5_5
6 または display.DIGITS_6_5

表示分解能設定の例：

次のコマンドを実行すると、SMU A の測定値表示が 5-1/2 桁分解能に設定されます。

```
display.smua.digits = display.DIGITS_5_5
```

メッセージ表示

注	大部分の display 関数とメッセージ表示に関連する属性値は自動的にユーザ画面を選択するように設計されています。画面表示をコントロールする属性値については「ディスプレイ画面」（p.14-3）も参照してください。 reset 関数（reset または smuX.reset）は既に定義されている表示メッセージとその構成には影響を与えませんが、表示モードをその前に使用していたソースメジャー表示モードへ戻します。

ユーザは自分が定義したメッセージを 260x 型のディスプレイに表示させることができます。たとえば、試験の進行中、260x. 型のディスプレイに次のようなメッセージを表示させておくことができます。

Test in Process
Do Not Disturb

上の行には最高 20 文字まで、下の行には最高 32 文字まで（いずれも空白文字を含む）を同時に表示させることができます。

> **注**　display.clear、display.setcursor、および display.settext 関数（いずれも以下の項で説明します）はオーバーラップ実行する非ブロック型コマンドです。つまり、スクリプトはこれらのコマンドの動作完了を待たずに次の動作を進めます。
>
> このような非ブロック型関数は性能上の考慮からディスプレイを即時には更新せず、まずバックグラウンドで処理を完了させておいてから処理時間に余裕が生ずるのを待って画面を更新します。

画面クリア

メッセージ表示のためにコマンドを送ったとしても、その前に定義していたメッセージは自動的にはクリアされません。新しいメッセージは現在表示されているメッセージの末尾に追加して書き込まれます。これを避けるには、新しいメッセージを表示する前に常に画面をクリアしてください。

一旦入力プロンプトメッセージを表示すると、オペレータが所定の動作を行った後もメッセージはそのまま画面に残ります。この場合も clear 関数を送って画面をクリアしてください。

次のコマンドはディスプレイの上下両方の行をクリアします。ただし表示ランプには影響を与えません。

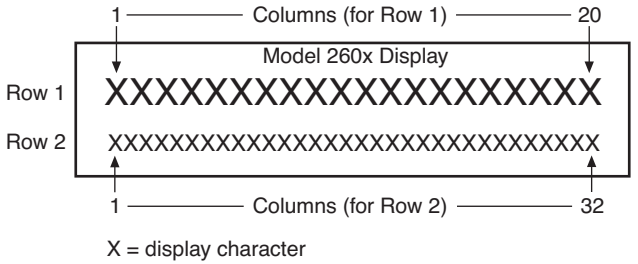
```
display.clear()
```

カーソル位置

メッセージを表示する際、その開始位置を決めるのがカーソルです。装置の起動時にはカーソル位置が Row 1、Column 1 に初期設定されます（図 14-1）。カーソルがこの位置にあるときにユーザ定義メッセージを表示すると上の行（Row 1）に表示されます。

上の行の表示テキストが行内に収まらなかったとしても下の行に自動的にラップされることはありません。現在の行に収まらなかった余分なテキストは切り捨てられます。テキストの最後の部分が切り捨てられた場合、カーソルは行の最後の桁に表示されます。

図 14-1
メッセージ表示の行 / 桁構成



カーソル位置を指定する関数は次の 2 通りの方法で使用します。

```
display.setcursor(row, column)
display.setcursor(row, column, style)
```

ここに、row = 1、2

```
column    = 1 ～ 20    (Row 1)
          = 1 ～ 32    (Row 2)
style     = 0          ( 非表示 )
          = 1          ( 点滅表示 )
```

style に 0 をセットするとカーソルは表示されません。1 をセットすると、カーソル位置にある文字が点滅表示することによってカーソルの位置を示します。

カーソルの現在位置を取得するには display.getcursor 関数を使用します。この関数は次の 3 通りの方法のいずれかで使用します。

```
row, column, style = display.getcursor()
row, column = display.getcursor()
row = display.getcursor()
```

例：次のコード例はまずカーソル位置を Row 2、Column 1 に設定し、続いてそのカーソル位置を読み取ります。

```
display.setcursor(2, 1)
row, column = display.getcursor()
print (row, column)
```

Output: 2.000000e+00 1.000000e+00

テキストメッセージの表示

メッセージを定義してそれを表示するには display.settext 関数を使用します。メッセージは現在のカーソル位置を起点として表示されます。

```
display.settext(text)
```

ここに、text は表示の対象となる文字列です。

例：次のコード例を実行すると、上の行に “Test in Process”、下の行に “Do Not Disturb” というメッセージが表示されます。

```
display.clear()
display.setcursor(1, 1, 0)
```

```
display.settext("Test in Process")
display.setcursor(2, 6, 0)
display.settext("Do Not Disturb")
```

文字コード

以下の特殊コードをテキスト文字列の中に埋め込んで、メッセージ書式の設定やカスタマイズに使用することができます。

$N	改行（Newline）- この文字以後のテキストを次の行に表示します。カーソルがすでに 2 行目の上にあった場合、‘$N’以後のテキストは無視されます。
$R	通常（Normal）モードでテキストを表示します。
$B	点滅（Blink）モードでテキストを表示します。
$D	低輝度（Dim）でテキストを表示します。
$F	テキストのバックグラウンドを点滅表示します。
$$	1 個の “$” を表示するエスケープシーケンス。

例：

次のコード例では、文字コード $N と #B を使用して “Test in Process” メッセージを上の行に通常モードで表示し、“Do Not Disturb” メッセージを下の行に点滅モードで表示します。

```
display.clear()
display.settext("Test in Process $N$BDo Not Disturb")
```

次のコード例は文字コード $$ を使用してメッセージ “You owe me $8” を上の行に表示します。

```
display.clear()
display.setcursor(1, 1)
display.settext("You owe me $$8")
```

$ を 1 個しか使用しなかった場合は $8 が未定義文字コードと見なされて無視されます。このため、このメッセージは “You owe me” と表示されてしまいます。

> **注**　特殊コードをテキスト文字列に埋め込むときには次のような注意が必要です。$ に続く文字もコードの一部であることを忘れてしまうことがあります。
> たとえば、上の行に “Hello” という文字列を表示して、下の行には “Nate” を表示するつもりで次のコマンドを送ったとします。
>
> ```
> display.settext("Hello$Nate")
> ```
>
> ところが、上のコマンドを実行すると上の行には “Hello”、下の行には “ate” と表示してしまいます。正しく表示させるには次の構文のテキストを送らなければなりません。
>
> ```
> display.settext("Hello$NNate")
> ```

テキストメッセージを返す

表示されているメッセージを返すには display.gettext 関数を使用します。この関数は次の 5 つのパターンのいずれかを使用します。

```
text = display.gettext()
text = display.gettext(embellished)
text = display.gettext(embellished, row)
text = display.gettext(embellished, row, column start)
text = display.gettext(embellished, row, column start, column end)
```

`embellished`	文字コードを含まない単純テキストを返すには false をセットします。文字コードを含めて返すには true をセットします。
`row`	1 または 2 をセットしてどちらの行を読み出すかを指定します。このパラメータを使用しなかった場合は両方の行を読み出します。
`column start`	テキストを読みだす最初の桁位置を指定します。
`column end`	テキストを読みだす最後の桁位置を指定します。

row パラメータを省略してコマンドを送ると、ディスプレイ上の両方の行を返します。このケースでは、上の行の終わりと下の行の始まりを区別するために文字コード $N が返り値に含まれています。たとえ embellished が false にセットされていたとしても文字コード $N は省略されません。

embellished が true にセットされていると、メッセージ作成時に文字列に含まれていた文字モードもすべて一緒に返されます。embellished を false にしておけばメッセージだけが返されます。

column start パラメータを省略してコマンドを送ると、Column 1 を指定したものと見なされ、column end を省略すると最後の桁（Row 1 は Column 20、Row 2 は Column 32）を指定したものとと見なされます。

入力プロンプト

メッセージ表示機能を前面パネルのコントロール要素と組み合わせることによりスクリプトの対話操作が可能になります。スクリプトを対話的に実行するためには、入力プロンプトを表示してオペレータが前面パネルのボタン類を使用して所定の操作ができるようにします。入力プロンプトを表示している間、試験は一時的に停止状態となってオペレータが前面パネルから所定の操作を行うのを待ちます。

メニュー

ユーザが定義したメニューをディスプレイに表示させることができます。メニューでは名前を表す menu 文字列が上の行に表示され、選択可能な items リストが下の行に表示されます。メニューを定義するには次の関数を使用します。

```
display.menu(menu, items)
```

ここに、 menu はメニューの名前を表す文字列（スペースを含めて最長 20 文字）です。items は 1 つ以上のメニュー項目を列挙した文字列であり、項目と項目の間をスペース文字で区切ります。

display.menu 関数を実行すると、スクリプトは一時動作を停止してオペレータがいずれかのメニュー項目を選択するのを待ちます。希望するメニュー項目の位置へ点滅カーソルを移させるにはホイールを回転させます。メニュー項目が画面に収まり切らない場合はホイールを右へ回転させると、今まで見えなかった項目が画面に現われます。カーソルが希望するメニュー項目上で点滅している状態で回転ホイール（または Enter キー）を押すことにより、その項目が選択されます。

メニューが表示されている間は EXIT キーを押してもスクリプトは中断されませんが、このキー操作によって nil が返されます。この操作に合わせてスクリプトを中断したい場合は、nil が返されたタイミングで exit 関数を呼び出してください。

例：次のコードは 2 種類のメニュー項目（Test1、Test2）をディスプレイに表示してオペレータにいずれかを選択させます。Test1 が選択されるとメッセージ “Running Test1” を表示し、Test2 が選択された場合はメッセージ “Running Test2” を表示します。

```
display.clear()
menu = display.menu("Sample Menu", "Test1 Test2")
if (menu == "Test1") then
  display.settext("Running Test1")
else display.settext("Running Test2")
end
```

パラメータ入力プロンプト

ユーザ画面上の現在のカーソル位置に編集可能な入力フィールドを作成するために次の 2 種類の関数が用意されています。display.inputvalue、display.prompt。

display.inputvalue 関数はユーザ画面の現在のカーソル位置を使用します。コマンドの処理が終了すると、ユーザ画面をこのコマンド実行直前の状態に戻します。一方、display.prompt 関数はユーザ画面をそのまま使用するのではなく、新たに編集可能な画面を作り出します。

この 2 種類の関数はそれぞれ 4 通りの方法で呼び出すことができます。

```
display.inputvalue(format)
display.inputvalue(format, default)
display.inputvalue(format, default, min)
display.inputvalue(format, default, min, max)

display.prompt(format, units, help)
display.prompt(format, units, help, default)
display.prompt(format, units, help, default, min)
display.prompt(format, units, help, default, min, max)
```

format – ここで指定する書式文字列に従ってユーザ画面の現在のカーソル位置に編集可能な入力フィールドが作成されます。入力フィールドの書式例：

```
+0.00 00  +00.0000E  +00  0.00000E+0
```

値フィールド：

+	正または負の入力を許すときに “+” 記号を含めます。“+” 記号を含めなかった場合は負の値を入力できなくなります。
0’s’	値の桁位置を定義します。最高 6 個までの ‘0’ を並べることができます（上の 3 番目と 4 番目の例参照）。
.	この記号を使用して、値が少数点（.）を必要とする位置にこの記号を挿入してください。

指数フィールド（省略可）

E	指数値を入力するときに “E” を含めます。
+	正または負の指数値入力を許すときに “+” 記号を含めます。“+” 記号を含めなかった場合は負の指数値を入力できなくなります。
0’s’	指数部の桁位置を定義します。

default – このオプションを使用すると、パラメータにデフォルト値が設定され、コマンド実行時に default で指定した値が画面に表示されます。

min および max – 入力フィールドに入力を許す最大値 / 最小値を設定するオプションです。ただし、値フィールドに “+” 記号を使用しなかった場合は、最小値リミットとして負の値を設定することはできません。“+” 記号を使用した場合は、負の最小値として負の値（たとえば -2）を指定することができます。

units および help – units はその値に使用する単位を指定するテキスト文字列です (8 文字以内)。たとえば、電圧ならば “V”、電流ならば “A” という具合に指定します。help は下の行に追加情報を表示するためのテキスト文字列です（32 文字以内)。

この 2 つの関数は編集可能な入力フィールドを表示するという点では似ていますが、display.inputvalue 関数は単位（units）とヘルプ情報（help）を表示する文字列を指定できません。

2 つの関数のいずれかが呼び出されると、スクリプトは一時的に実行を停止してオペレータがソースレベルを入力するのを待ちます。オペレータが値を入力して回転ホイールまたは Enter キーを押すと、プログラムは実行を再開します。

例：

次のコードはオペレータにソース値の入力を要求します。

```
display.clear()
value = display.prompt("0.00", "V", "Enter source voltage")
display.screen = display.SMUA
smua.source.levelv = value
```

スクリプトはプロンプトメッセージが画面に表示してオペレータが電圧レベルの値を入力するのを待ちます。値を入力されると、ディスプレイを SMU A のソースメジャー画面に切換えて、ソースレベルに入力された値をセットします。

注 オペレータがソース値を入力する代わりに EXIT キーを押すと、value には nil がセットされます。

上で説明したコードの 2 行目は別な入力フィールド関数に置き換えが可能です。

```
value = display.inputvalue("0.00")
```

両者の違いは、置き換え後の関数が単位（“V”）と “Enter source value”メッセージを画面に表示しないことだけです。

表示ランプ

現在どの表示ランプが点灯しているのかを知るには次のコードを送ります。

```
annun = display.getannunciators()
print(annun)
```

このコードは点灯状態を 16 ビットの 2 進数表現に変換したビットマップを返します。ビットマップのそれぞれのビットが 1 つの表示ランプに対応します。あるビットが “1”であれば対応する表示ランプは点灯しており、そのビットが “0”にセットされていれば表示ランプは消灯しています。

ビット位置と表示ランプの対応については表 14-2 をご覧ください。実際に返される値はセットされているビットに桁により重み付けを行って加算した総和です。表には各ビットに重み付けをした値も記載されています。

たとえば、返り値が 34061 であったとすれば、これに対応する 2 進数は次のように表現されます。

1000010100001101

この 2 進数表現では、ビット 16、11、9、4、3、および 1 が “1” にセットされています。これを表 14-2 と対照することにより、REL、REM、EDIT、AUTO、4W、および EDIT ランプが点灯していることが分ります。

表 14-2
ビット位置と表示ランプの対応

ビット	16	15	14	13	12	11	10	9
表示ランプ	REL	REAR	SRQ	LSTN	TALK	REM	ERR	EDIT
対応する 10 進数値*	32768	16384	8192	4096	2048	1024	512	256
2 進数値	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1

ビット	8	7	6	5	4	3	2	1
表示ランプ	SMPL	STAR	TRIG	ARM	AUTO	4W	MATH	FILT
対応する 10 進数値*	128	64	32	16	8	4	2	1
2 進数値	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1

* 対応する 10 進数値はそのビットが “1” にセットされているときの値です。ビットが “0” にセットされているときは値を持ちません。

表 14-2 に記載の表示ランプの一部は 260x 型では使用されていません。

LOCAL ロックアウト

前面パネルの LOCAL キーはリモート操作をキャンセルしてコントロールを前面パネルに戻すときに使用します。ただし、試験の不用意な中断を予防するために LOCAL キーはロックアウトされていることがあります。ロックされている LOCAL キーは NO-OP（動作不能）状態になっています。LOCAL キーをロック、またはロックを解除するには以下の属性値を使用します。

```
display.locallockout = lockout
```

lockout には次のいずれかの値を設定します。

0 または display.UNLOCK
1 または display.LOCK

LOCAL ロックアウトの例：

次のコマンドは LOCAL キーをロックアウトします。

```
display.locallockout = display.LOCK
```

テストメニューのローディング

LOAD TEST メニューには前面パネルから実行可能なスクリプト試験（USER、FACTORY）が項目として表示されます。ファクトリスクリプト試験（機能試験）はローディングされて不揮発性メモリに保存された状態で工場から出荷されます。これらの機能は FACTORY TESTS サブメニューから呼び出します。

ユーザスクリプトを 260x 型にローディングしても、前面パネルの USER TESTS サブメニューに自動的に登録される訳ではありません。メニュー名とチャンクはユーザ自身がメニューに追加しなければなりません。

ユーザスクリプトの保存

260x 型ソースメータへローディング後のユーザスクリプトは、不揮発性メモリへ保存することも可能です。不揮発性メモリに保存しなかったスクリプトは、260x 型の電源が切れると同時に失われます。

スクリプトを Test Script Builder からローディングするのであれば、スクリプトが不揮発性メモリに保存されるようにランチャーを設定することができます（「Test Script Builder を使用する」、p.2-12、参照）。

別なプログラムを使用してユーザスクリプトをローディングする場合は、myscript.save() を用いてスクリプトを不揮発性メモリに保存します（「ユーザスクリプトの保存」、p.2-43、参照）。

USER TESTS メニューの項目追加

次の関数を使用して USER TESTS サブメニューに項目を追加します。この関数は次の 2 通りの方法で使用します。

```
display.loadmenu.add(displayname, chunk)
display.loadmenu.add(displayname, chunk, memory)
```

displayname	メニューに追加する項目名を表す文字列
chunk	実行するコード
memory	chunk と displayname を不揮発性メモリに保存するか否かを指定

memory に次のいずれかの値をセットします。
0 または display.DONT_SAVE
1 または display.SAVE

memory のデフォルト設定は display.SAVE です。

chunk はスクリプト、関数、変数、およびコマンドによって構成されています。memory に display.SAVE をセットしておくと、コマンドがチャンクと共に不揮発性メモリに保存されます。ただし、このチャンクが使用するスクリプト、関数、および変数は display.SAVE を指定しても保存されません。関数と変数はスクリプトと共に保存する必要があります（「ユーザスクリプトの保存」、参照）。不揮発性メモリに保存されなかったスクリプトは 260x 型の電源が切れると同時に失われます。次に示す例 1 を参照。

例 1：

“DUT1”という名前の関数を含むスクリプトがすでに 260x 型へローディングされており、スクリプトは不揮発性メモリに保存されていないものとします。

ここで、“Test”という名前の試験を USER TESTS メニューに追加したいとします。この試験は関数 “DUT1”を実行してブザーを鳴動させるものです。以下のコマンドはメニューに項目 “Test”を追加し、チャンクを定義し、続いて displayname と chunk を不揮発性メモリに保存します。

```
display.loadmenu.add(“Test”, “DUT1() beeper.beep(2, 500)”,
display.SAVE)
```

前面パネルの USER TESTS メニューから “Test”を実行すると、“DUT1”という名前の関数が実行されてブザーを 2 秒間鳴らします。

ここで、一旦 260x 型の電源を切ってから再投入するものとします。このスクリプトは不揮発性メモリに保存されていませんでしたから、関数 “DUT1”はこの操作で失われます。再び前面パネルから “Test”を実行すると、ブザーは鳴りますが、“DUT1”はすでにチャンクに存在しませんから実行されません。

例 2：

次に示すコマンドは前面パネルの “USER TESTS”サブメニューにチャンク“testpart([[Part1]], 5.0)”に対応する “Part1”という項目を追加し、それを不揮発性メモリに保存します。

```
display.loadmenu.add("Part1", "testpart([[Part1]], 5.0)",
display.SAVE)
```

USER TESTS メニューの項目削除

前面パネルの USER TESTS サブメニューからある項目を削除したい場合には次の関数を使用します。

```
display.loadmenu.delete(displayname)
```

`displayname` メニューから削除する項目名を表す文字列

例 ：

次のコマンドを実行すると、“Part1”という名前の項目が前面パネルの USER TESTS サブメニューから削除されます。

```
display.loadmenu.delete("Part1")
```

前面パネルからテストを実行

前面パネルに表示されたユーザテスト、ファクトリテストを実行する手順は次の通りです。

1. LOAD キーを押して LOAD TEST メニューを表示させます。
2. USER または FACTORY メニュー項目を選択します。
3. 点滅カーソルを実行したいテスト項目へ移動させます。
4. RUN キーを押してテストを実行します。

ディスプレイトリガ

スクリプトの実行を一定時間停止させ、オペレータが TRIG キーを押すのを待ってから実行を継続するという使い方をすることができます。この方法では、TRIG キーが押されるか、または指定した停止時間が終了することによってスクリプトの実行が再開されます。

ディスプレイ画面からトリガを発生する方法で使用できるのは次の 2 つの関数です。

```
display.trigger.wait(timeout)
display.trigger.clear()
```

ここに、timeout は秒単位で指定したタイムアウト時間です。

TRIG キーを押すと、前面パネルのトリガディテクタがセットされます。このディテクタを有効に使用するためには次回 display.trigger.wait 関数を呼び出す前にクリアしておかなければなりません。ディテクタがセットされた状態でトリガ待ちコマンドを送ることによって、トリガが即時検出されてスクリプトが実行を再開します。トリガディテクタをクリアするには display.trigger.clear 関数を送ってください。

TRIG キーとタイムアウトのどちらによってトリガが検出されたのかを知りたい場合は、トリガ待ち関数が返す値をチェックします。

```
triggered = display.trigger.wait(timeout)
print(triggered)
```

Output: `1.000000e+00` (TRIG キーが押された)
`0.000000e+00` (タイムアウト時間終了)

ディスプレイトリガの例 :

次に示すコードではスクリプトの実行を一時停止して TRIG キープレスを要求するプロンプトを表示し、オペレータの準備が整ったときに TRIG キーを押してもらいます。10 分（600 秒）を経過しても TRIG キーが押されなかった場合は自動的に実行を継続します。

```
display.clear()
display.setcursor(1, 1, 0)
display.settext("Take a Break")
display.setcursor(2, 1, 0)
display.settext("Press TRIG to continue")
display.trigger.wait(600)
display.trigger.clear()
display.clear()
```

トリガ検出条件（TRIG キープレス、またはタイムアウト）が満たされると、トリガディテクタとディスプレイのクリアを行います。

キープレスコード

キーコードを送る

リモートからキーコードを送ることによって前面パネルキーや回転ノブ「プレス」を模擬することができます。ノブの左右の回転（1 クリックごと）についてもキーコードを送って模擬することができます。このような操作に使用できるのが display.sendkey 関数です。

```
display.sendkey(keycode)
```

ここに、keycode は前面パネルのコントロール要素に対応する値です。
それぞれのコントロール要素に対応するキーコードを表 14-3 に示します（アルファベット順）。

キープレスの例：

次のコマンドのどちらも MENU キーを押したのと同じ動作を行います。

```
display.sendkey(display.KEY_MENU)
display.sendkey(68)
```

表 14-3
display.sendkey が送るキーコード一覧

display.KEY_AUTO	or	73	display.KEY_OUTPUTA	or	88
display.KEY_CONFIG	or	80	display.KEY_OUTPUTB	or	96
display.KEY_DIGITSA	or	87	display.KEY_RANGEDOWN	or	81
display.KEY_DIGITSB	or	84	display.KEY_RANGEUP	or	65
display.KEY_DISPLAY	or	72	display.KEY_RECALL	or	85
display.KEY_ENTER	or	82	display.KEY_RELA	or	70
display.KEY_EXIT	or	75	display.KEY_RELB	or	67
display.KEY_FILTERA	or	77	display.KEY_RIGHT	or	103
display.KEY_FILTERB	or	74	display.KEY_RUN	or	71
display.KEY_LEFT	or	104	display.KEY_SPEEDA	or	94
display.KEY_LIMITA	or	93	display.KEY_SPEEDB	or	91
display.KEY_LIMITB	or	90	display.KEY_SRCA	or	79
display.KEY_LOAD	or	95	display.KEY_SRCB	or	76
display.KEY_MEASA	or	86	display.KEY_STORE	or	78
display.KEY_MEASB	or	83	display.KEY_TRIG	or	92
display.KEY_MENU	or	68	display.WHEEL_ENTER	or	97
display.KEY_MODEA	or	69	display.WHEEL_LEFT	or	107
display.KEY_MODEB	or	66	display.WHEEL_RIGHT	or	114

キープレスコードの取り込み

260x 型は最後にどの前面パネルキーが押されたのかをキーコードとして記憶しています。装置起動時（またはローカルからリモートモードへの移行時）に、このキーコードが 0（display.KEY_NONE）にリセットされます。

前面パネルキーが押されたときに、そのキーコードを取り込んで返すことも可能です。キープレスコードを取り込んで返す操作に使用できるのは display.getlastkey と display.waitkey の 2 つの関数です。

display.getlastkey

最後に押されたキーのキーコードを即時取得したい場合は display.getlastkey 関数を使用します。

```
key = display.getlastkey()
print(key)
```

上記のコードはキーコード値（表 14-4 参照）を返します。返り値が 0（display.KEY_NONE）はキーコード履歴がクリアされたことを意味します。

表 14-4
display.getlastkey が返すキーコード値

0 (display.KEY_NONE)	82 (display.KEY_ENTER)
65 (display.KEY_RANGEUP)	83 (display.KEY_MEASB)
67 (display.KEY_RELB)	84 (display.KEY_DIGITSB)
68 (display.KEY_MENU)	85 (display.KEY_RECALL)
69 (display.KEY_MODEA)	86 (display.KEY_MEASA)
70 (display.KEY_RELA)	87 (display.KEY_DIGITSA)
71 (display.KEY_RUN)	90 (display.KEY_LIMITB)
72 (display.KEY_DISPLAY)	91 (display.KEY_SPEEDB)
73 (display.KEY_AUTO)	92 (display.KEY_TRIG)
74 (display.KEY_FILTERB)	93 (display.KEY_LIMITA)
75 (display.KEY_EXIT)	94 (display.KEY_SPEEDA)
76 (display.KEY_SRCB)	95 (display.KEY_LOAD)
77 (display.KEY_FILTERA)	97 (display.WHEEL_ENTER)
78 (display.KEY_STORE)	103 (display.KEY_RIGHT)
79 (display.KEY_SRCA)	104 (display.KEY_LEFT)
80 (display.KEY_CONFIG)	114 (display.WHEEL_RIGHT)
81 (display.KEY_RANGEDOWN)	

注　SMU A と SMU B の OUTPUT ON/OFF キー操作をこの関数で追跡することはできません。

display.waitkey

キーが次回押されるのを待ち構えてキーコードを取得するには display.waitkey 関数を使用します。

```
key = display.waitkey()
```

display.waitkey 関数を送った後、スクリプトは一旦実行を休止してオペレータが前面パネルのキーを押すのを待ちます。たとえば MEAS キーが押されたとすれば、このキーに対応するキーコードである 86 という値を返します。キーコードの値については表 14-3 を参照してください。

例：次のコードはプロンプトを表示して、EXIT キーを押してスクリプトを中断するか、またはそれ以外のキーを押してスクリプトを継続するかの判断を要求します。

```
display.clear()
display.setcursor(1, 1)
display.settext("Press EXIT to Abort")
display.setcursor(2, 1)
display.settext("or any key to continue")
key = display.waitkey()
display.clear()
display.setcursor(1, 1)
if (key == 75) then
  display.settext("Test Aborted")
  exit()
else display.settext("Test Continuing")
end
```

上記のコードはオペレータがキーを押すのを待ち構えてそのキーコードを捉えます。EXIT キーに対応するキーコード値は 75 です。EXIT が押されるとスクリプトを終了し、それ以外のキーが押された場合はスクリプトを継続実行します。

15
性能確認

セクション 15 のトピックス

はじめに , 15-2

確認試験の要件 , 15-2
- 環境条件 , 15-2
- ウォームアップ時間 , 15-3
- 電源ライン , 15-3
- 推奨試験装置 , 15-3
- 性能確認の限界値 , 15-4

ファクトリデフォルトの呼び戻し , 15-5

確認試験手順の実行 , 15-5
- 試験のまとめ , 15-5
- 試験実施のための考慮事項 , 15-5
- ソースレンジと出力値の設定 , 15-6
- 測定レンジの設定 , 15-6

出力電圧の確度 , 15-7

電圧測定の確度 , 15-8

出力電流の確度 , 15-9

電流測定の確度 , 15-11

はじめに

このセクションで解説する操作方法を用いて 260x 型が装置の確度仕様（1 年間保証）に記載の要件を満たしているか否かを確認します。たとえば次のような目的でこの確認を行います。

- 装置受領時に輸送中の損傷の有無を確認するため。
- 装置が工場仕様を満足していることを確認するため。
- 校正が必要であるか否かを確認するため。
- 校正作業後に、校正が適正に行われたか否かをチェックするため。

警告 **このセクションに記載された情報は専門のトレーニングを受けてサービスを担当される技術者を対象とするものです。それ以外の方はこのセクションに記載の手順を実行しないでください。作業される方を危険な電圧に曝す作業内容が一部に含まれており、感電事故や、場合によっては致命的な傷害を発生させる恐れがあります。危険な電圧を取り扱う作業では必ず適切な安全対策を施してください。**

注 *保証期間内の装置であるにも拘わらず性能仕様を満たしていないと思われる場合は、Keithley の代理店または直接工場へ連絡して是正措置について問い合わせてください。*

確認試験の要件

性能確認試験を実施するときは必ず以下の条件を満たしてください。

- 適正な環境条件下で試験を行う。
- 指定されたウォームアップ時間が経過してから試験を行う。
- 正しい電源電圧を使用する。
- 適切な試験装置を使用する。
- 指定された出力信号と読み取り値リミットを使用する。

環境条件

以下の条件に適合する試験環境を準備して性能確認を行ってください。

- 周囲温度：18 ～ 28°C (65 ～ 82°F)
- 相対湿度 70% 未満（特に指定のある場合を除く）

ウォームアップ時間

260x 型の確認試験操作を開始する前に少なくとも 2 時間以上のウォームアップ時間を置いてください。装置が極端な温度条件（前ページで指定した温度範囲外）に曝されていた場合は、この時間を更に長くして装置の内部温度を安定化させる必要があります。標準的には指定温度範囲を 10 ℃（18°F）超えるごとに安定化時間を 1 時間長くしてください。

試験に使用する他の機器についても、少なくともそれぞれのメーカーが指定する最小ウォームアップ時間を置いてください。

電源ライン

260x 型は電圧が 100 ～ 240V、周波数が 50Hz または 60 Hz の電源を必要とします。必ずこの条件を満たす電源を使用して確認試験を行ってください。

推奨試験装置

推奨する試験装置をまとめて表 15-1 に示します。表 15-1 に示す仕様を満たすか、あるいはそれ以上の性能を持つ装置であれば、ここに列挙した以外の装置を使用してもかまいません。試験装置自体が持つ不確定性が試験結果に加算されることを常に考えておく必要があります。この観点から、一般的には 260x 型の仕様よりも 4 倍以上優れた確度を持つ試験装置を使用するべきです。表 15-1 にはそれぞれの推奨試験装置の持つ不確定性も記載されています。

表 15-1
推奨試験装置

機種	メーカー / モデル名	確度		
デジタルマルチメータ	Agilent 3458A	DC 電圧 [1]	90mV: 0.9V: 5.4V: 36V:	±8ppm ±5ppm ±8ppm ±11ppm
		DC 電流 [1]	90nA: 0.9µA: 9µA: 90µA: 0.9mA: 9mA: 90mA: 0.9A:	±430ppm ±55ppm ±25ppm ±23ppm ±20ppm ±20ppm ±35ppm ±110ppm
0.5Ω、250W、0.1% 精密抵抗	Isotek RUG-Z-R500-0.1-TK3	抵抗 [2]	0.5Ω:	±125ppm

1. 推奨モデルが指定のポイント測定のために持つべき確度が 90 日仕様に記載されています。
2. 3A レンジでのテストに用いる抵抗については、実際の使用前にデジタルマルチメータの抵抗測定機能を使用して特性・確度を把握しておく必要があります。

性能確認の限界値

性能確認限界値についてこのセクションが提示する値は 260x 型の 1 年確度仕様のみを使用して計算された値であり、試験に使用する装置自体が持つ不確かさは考慮されていません。したがって、もし測定値が許容範囲を超えた場合には、260x 型の仕様と試験装置仕様の両方を考慮に入れた限界値を新たに計算する必要があります。

限界値計算の例

性能確認限界値を計算する例として、出力値 5.4V を使用して 6V DC 出力レンジの試験を行うケースを考えます。260x 型の 5.4V DC 出力に適用される 1 年確度仕様である「± (出力の 0.02% + 1.8mV オフセット)」から、出力限界値は次のように計算されます。

出力限界値 = 5.4V ± [(5.4V × 0.02%) + 1.8mV]
出力限界値 = 5.4V ± (0.00108 + 0.0018)
出力限界値 = 5.4V ± 0.00288V
出力限界値 = 5.39712V ～ 5.40288V

ファクトリデフォルトの呼び戻し

性能確認の操作を実行する前に、次の手順に従って装置を工場出荷時の前面パネル操作（ベンチ）デフォルトの状態に戻してください。

1. MENU キーを押します。装置のディスプレイに次のプロンプトが表示されます。
   MAIN MENU
   SAVE-SETUP COMMUNICATION TEST
2. SAVE-SETUP を選択し、続いて ENTER を押します。ディスプレイには次のメッセージが表示されます。
   SAVE SETUP MENU
   SAVE RECALL POWERON
3. RECALL を選択して ENTER を押します。ディスプレイに次のメッセージが表示されます。
   RECALL SETUP
   FACTORY USER-1 USER-2 USER-3 USER-4 USER-5
4. FACTORY を選択して ENTER を押すことによりデフォルト設定が呼び戻されます。

確認試験手順の実行

試験のまとめ

- DC 電圧出力確度
- DC 電圧測定確度
- DC 電流出力確度
- DC 電流測定確度

260x 型の性能が仕様の範囲に収まらず、かつ保証期間がすでに切れている場合は装置校正を説明したセクション 16 の説明をご覧ください。

試験実施のための考慮事項

確認試験実施にあたって以下の項目に注意してください。

- 工場出荷時の前面パネルデフォルトに戻されていることを確認します（上記の説明参照）。
- 試験装置が適正にウォームアップされており、260x 型の出力端子に正しく接続されていることを確認します（電圧測定には 4 線式センシングを使用）。
- 260x 型のソースレンジが正しく設定されていることを確認します。
- 測定前に、260x 型の出力が ON になっていることを確認します。
- 試験装置側の機能とレンジが正しく設定されていることを確認します。
- 測定の前に充分な整定時間を置いて 260x 型の出力信号を安定化させます。
- 試験装置と 260x 型は直接接続し、接続の途中にスキャナやマルチプレクサ、その他のスイッチング装置を挿入しないでください。

警告 コモンモード電圧（LO と筐体 GND 間の電圧）の許容最大値は 250 VDC です。これを超える電圧の印加は絶縁破壊を引き起こして、感電事故発生の原因になります。

260x 型の端子は設置カテゴリー I の回路との接続に適合する定格で設計されています。したがって、260x 型の端子を CAT II、CAT III、CAT IV 回路には接続しないでください。ソースメータの端子を CAT I 規格を超える回路に接続すると装置の損傷を招き、オペレータを危険な電圧に曝す恐れがあります。

出力端子とガード端子には危険な高電圧が印加される可能性があります。感電事故防止のため、装置が通電状態のときには絶対に 260x 型の接続や接続切り離し作業を行わないでください。場合によっては致命的な傷害を引き起こす可能性があります。出力端子に接続されているケーブルを処理するときは、必ず事前に前面パネルから電源を切るか、または 260x 型背面の電源ケーブルを引抜いてください。装置をスタンバイ状態に入れたとしても、ハードウェアやソフトウェア等の障害によって出力が完全にオフになっていない可能性がありますのでご注意ください。

ソースレンジと出力値の設定

確認の必要な項目の試験を開始する前に、次の説明に従ってソースレンジと出力値を正しく設定してください。

1. SRC キーを押して正しいソース機能を選択します。
2. 回転ノブまたは ENTER キーを押して編集モードに入ります（EDIT ランプ点灯）。
3. カーソルがソース表示フィールド上で点滅している状態で、ソースレンジを現在出力している値に対して可能な最低レンジに設定します。具体的には 上下の RANGE キーを使用して値を設定します。たとえば、ソース値として 5.4V を出力するのであれば、ソースレンジとしては 6V を選択してください。ソース値として 5.4V、レンジとして 6V を選択したとすれば、2602 型のチャンネル A のソースフィールドには次の内容が表示されるはずです。

   SrcA:+5.40000 V

4. 回転ノブと CURSOR キーを使用してソース値を必要な値に設定し、ENTER または回転ノブを押して設定作業を終了します。

測定レンジの設定

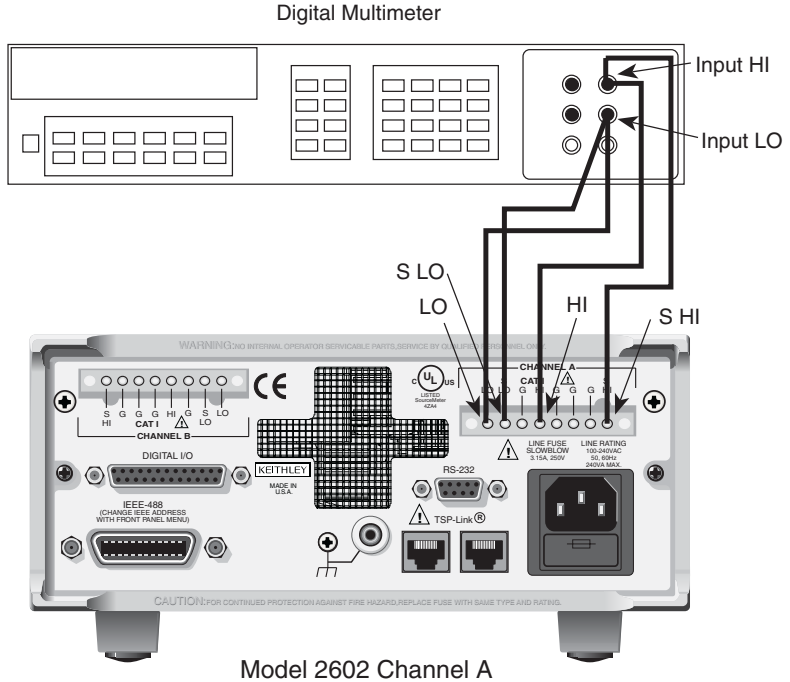
電流または電圧のソース出力と測定を同時に行う場合は、測定レンジがソースレンジと連動しますから測定レンジだけを独立にコントロールすることはできません。したがって、電圧または電流の測定確度を試験するケースでは、ユーザがレンジを設定する必要はありません。

出力電圧の確度

以下のステップにしたがって、260x 型の出力電圧確度が仕様限界値の範囲内に収まっていることを確認してください。この試験では各フルレンジ値に対応する出力電圧を設定し、その電圧を精密デジタルマルチメータで測定します。

1. 装置の電源を切り、デジタルマルチメータを 260x 型の出力端子に接続します（4 線式接続、図 15-1 参照）。
2. マルチメータの DC 電圧測定機能を選択します。
3. 2602 型の場合はシングルチャンネル表示モードを選択します。260x 型の SRC キーを押して電圧ソース機能を選択し、この段階でソース出力が ON になっていることを確認します。
4. 次の順で操作して 260x 型を 4 線（リモートセンス）モードに入れます。CONFIG、次に SRC を押し、続いて V-SOURCE > SENSE-MODE > 4-WIRE の順に選択。
5. 表 15-2 に記載の各電圧値における出力電圧確度を確認します。具体的には各試験ポイントについて以下の操作を行います。
   - 正しいソースレンジを選択
   - 260x 型の出力電圧を指示された値に設定
   - マルチメータ指示値が表に示された限界値の範囲内であることを確認

図 15-1
電圧確度確認のための接続



Model 2602 Channel A
Connections Shown

6. 表 15-2 に記載の値と絶対値が同じで極性が反対の出力電圧についても上記で説明した手順を繰り返します。
7. 2602 型の場合は、もう一方のチャンネルについても同じ操作を行います。

表 15-2
出力電圧の確度限界値

260x 型 ソースレンジ	260x 型出力 電圧設定	出力電圧限界値 (1 年、18 °C ～ 28°C)
100mV	90.000mV	89.732 ～ 90.268mV
1V	0.90000V	0.89942 ～ 0.90058V
6V	5.4000V	5.39712 ～ 5.40288V
40V	36.000V	35.9808 ～ 36.0192V

電圧測定の確度

以下のステップにしたがって、260x 型の電圧測定確度が仕様限界値の範囲内に収まっていることを確認してください。この試験ではソース電圧を設定して、その値を精密デジタルマルチメータで測定し、260x 型の電圧読み取り値が要求される限界値の範囲内に収まっていることを確認します。

1. 装置の電源を切り、デジタルマルチメータを 260x 型の出力端子に接続します（4 線式接続、図 15-1 参照）。
2. マルチメータの DC 電圧機能を選択します。
3. 2602 型の場合はシングルチャンネル表示モードを選択します。
4. 次の順で操作して 260x 型を 4 線（リモートセンス）モードに入れます。CONFIG、次に MEAS を押し、続いて V-MEAS > SENSE-MODE > 4-WIRE の順に選択。
5. SRC キーと MEAS キーを押して 260x 型を電圧ソースと電圧測定モードの両方に入れ、ソース出力が ON になっていることを確認します。
6. 表 15-3 に記載の各電圧値における電圧測定値確度を確認します。具体的には各試験ポイントについて以下の操作を行います。
   - 正しいソースレンジを選択
   - 260x 型の出力電圧をデジタルマルチメータで測定する値に合わせて設定
   - 必要ならば TRIG キーを押して読み取り値を表示
   - 260x 型の電圧読み取り値が表に指定された限界値の範囲内に収まっていることを確認する。電圧ソースを必要な値に設定できない場合もあり得ます。その場合は可能な最も近い値に設定し、それに応じて読み取り値の限界値も修正してください。
7. 表 15-3 に記載の値と絶対値が同じで、極性が反対のソース電圧についても上記で説明した手順を反復します。
8. 2602 型の場合は、もう一方のチャンネルについても同じ操作を行います。

表 15-3
電圧測定の確度限界値

260x 型のソースと測定レンジ[1]	ソース電圧[2]	260x 型の電圧読み取り限界値（1 年、18°C ～ 28°C)
100mV	90.000mV	89.8365 ～ 90.1635mV
1V	0.90000V	0.899665 ～ 0.900335V
6V	5.4000V	5.39819 ～ 5.40181V
40V	36.000V	35.9866 ～ 36.0134V

1. 電圧ソースと電圧測定の両機能を同時に使用する場合、測定レンジはソースレンジに連動します。
2. 精密デジタルメータで測定した値。厳密に一致しない場合は可能な最も近い値を用い、読み取り値の限界値も変化に合わせて修正します。

出力電流の確度

以下のステップにしたがって、260x 型の出力電流確度が仕様限界値の範囲内に収まっていることを確認してください。

1. 装置の電源を切り、デジタルマルチメータを 260x 型の出力端子に接続します（図 15-2 参照）。
2. マルチメータの DC 電流測定機能を選択します。
3. 2602 型の場合はシングルチャンネル表示モードを選択します。
4. 260x 型の SRC キーを押して電流ソース機能を選択し、この段階でソース出力が ON になっていることを確認します。
5. 100nA から 1A までの各レンジ（表 15-4 参照）の電流値について出力電流の確度を確認します。具体的には各試験ポイントについて以下の操作を行います。
   - 正しいソースレンジを選択
   - 260x 型の出力電流を指示された値に設定
   - マルチメータ指示値が表に示された限界値の範囲内であることを確認
6. 表 15-4 に記載の値と絶対値が同じで、極性が反対の出力電流についても上記で説明した手順を反復します。
7. 出力を OFF に戻して接続を図 15-3 に示す状態へと変更します（図のように 4 線式接続を用いて 0.5Ω 抵抗を接続します）。
8. DMM の DC 電圧機能を選択します。
9. 3A レンジを選択してステップ 4 から 7 までを繰り返します。DMM の電圧指示値と予め厳密な値の分っている 0.5Ω 抵抗とから電流値を計算（I=V/R）します。
10. 2602 型の場合は、もう一方のチャンネルについても同じ操作を行います。

図 15-2
電流確度確認のための接続（100nA ～ 1A レンジ）

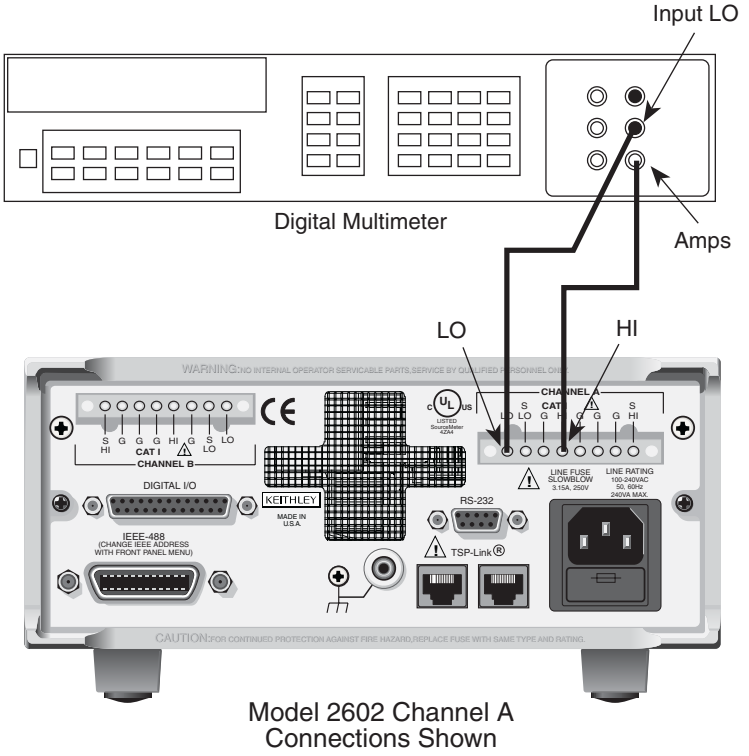
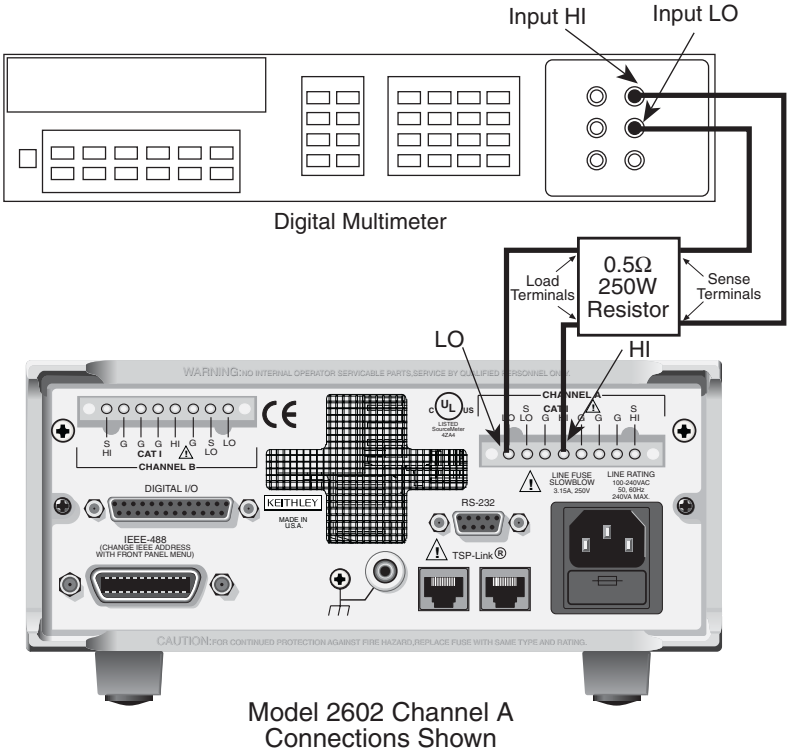


表 15-4
出力電流の確度限界値

260x 型 ソースレンジ	260x 型 出力電流設定	出力電流限界値 (1 年、18°C ～ 28°C)
100nA	90.000nA	89.846 ～ 90.154nA
1µA	0.90000µA	0.89913 ～ 0.90087µA
10µA	9.0000µA	8.9953 ～ 9.0047µA
100µA	90.000µA	89.943 ～ 90.057µA
1mA	0.90000mA	0.89953 ～ 0.90047mA
10mA	9.0000mA	8.9943 ～ 9.0057mA
100mA	90.000mA	89.953 ～ 90.047mA
1A	0.90000A	0.89885 ～ 0.90115A
3A	2.40000A	2.39706 ～ 2.40294A

図 15-3
電流確度確認のための接続（3A レンジ）



電流測定の確度

以下のステップにしたがって、260x 型の電流測定確度が仕様限界値の範囲内に収まっていることを確認してください。この操作では 260x 型を電流ソースとして機能させて正確な電流を供給し、その電流が 260x 型で測定した電流値が課せられた限界値の範囲内に収まっていることを確認します。

1. 装置の電源を切り、デジタルマルチメータを 260x 型の端子に接続します（図 15-2 参照）。
2. マルチメータの DC 電流機能を選択します。
3. 2602 型の場合はシングルチャンネル表示モードを選択します。
4. SRC キーと MEAS キーを押して 260x 型を電流ソースと電流測定モードの両方に入れ、ソース出力が ON になっていることを確認します。

5. 表 15-5 に記載の各電流値における電流測定値確度を確認します。具体的には各測定ごとに以下の操作を行います。
   - 正しいソースレンジを選択
   - 260x 型のソース出力を、デジタルマルチメータで測定する値に合わせて正しく設定
   - 必要ならば TRIG キーを押して読み取り値を表示
   - 260x 型の電流読み取り値が表に指定された限界値の範囲内に収まっていることを確認する。電流ソースを必要な値に正確に設定できない場合もあり得ます。その場合は可能な最も近い値に設定し、それに応じて読み取り値の限界値も修正してください。
6. 表 15-5 に記載の値と絶対値が同じで、極性が反対の校正電流についても上記で説明した手順を反復します。
7. 出力を OFF にして接続を図 15-3 に示す状態に変更し、DMM の機能を電圧測定に切換えます。
8. 3A レンジを選択してステップ 4 から 6 までを繰り返します。DMM の電圧指示値と予め厳密な値の分っている 0.5Ω 抵抗とから電流値を計算します。
9. 2602 型の場合は、もう一方のチャンネルについても同じ操作を行います。

表 15-5
電流測定の確度限界値

260x 型のソースと測定レンジ[1]	ソース電流[2]	260x 型の電流読み取り限界値 (1 年、18°C ～ 28°C)
100nA	90.000nA	89.855 ～ 90.145nA
1µA	0.9000µA	0.899475 ～ 0.900525µA
10µA	9.0000µA	8.99715 ～ 9.00285µA
100µA	90.000µA	89.973 ～ 90.027µA
1mA	0.9000mA	0.89976 ～ 0.90024mA
10mA	9.0000mA	8.9973 ～ 9.0027mA
100mA	90.000mA	89.976 ～ 90.024mA
1A	0.90000A	0.89923 ～ 0.90077A
3A	2.4000A	2.3981 ～ 2.4019A

1. 電流ソースと電流測定の両機能を同時に使用する場合、測定レンジはソースレンジに連動します。
2. 精密デジタルメータで測定した値。厳密に一致しない場合は可能な最も近い値を用い、読み取り値の限界値も変化に合わせて修正します。

16
校正

セクション 16 のトピックス

はじめに , 16-2

環境条件 , 16-2
温度と相対湿度 , 16-2
ウォームアップ時間 , 16-2
供給電源 , 16-2

校正作業の注意事項 , 16-3
校正サイクル , 16-3
校正に使用できる推奨装置 , 16-4
校正エラー , 16-5

校正 , 16-5
校正ステップ , 16-5
校正コマンド , 16-6
校正手順 , 16-8

はじめに

このセクションで説明する操作手順にしたがって 260x 型の校正を行います。この作業を実施するには DC 電圧と電流を測定する精密試験装置が必要となります。

警告 **このセクションに記載された情報は専門のトレーニングを受けてサービスを担当される技術者を対象とするものです。これらの操作の中には危険な電圧に曝される恐れのある作業も含まれますから、有資格者以外はこのセクションで説明する作業を行わないでください。**

環境条件

温度と相対湿度

特に別段の指定がない限りは、周囲温度 18 ～ 28° C（65 ～ 82°F)、相対湿度 70% 未満の環境で校正作業を行ってください。

ウォームアップ時間

260x 型の校正作業を開始する前に少なくとも 2 時間以上のウォームアップ時間を置いてください。

装置が極端な温度条件（上記に指定した温度範囲外）に曝されていた場合は、この時間を更に長くして装置の内部温度を安定化させる必要があります。標準的には指定温度範囲を 10 ℃（18°F）超えるごとに安定化時間を 1 時間長くしてください。

試験に使用する他の機器についても、少なくともそれぞれのメーカーが指定する最小ウォームアップ時間を置いてください。

供給電源

260x 型は電圧が 100 ～ 240V、周波数が 50Hz または 60 Hz の電源を必要とします。校正には必ずこの条件を満たす電源を用いてください。

校正作業の注意事項

校正操作実施にあたって以下の項目に注意してください。

- 試験装置が適正な時間ウォームアップをされており、260x 型の端子に正しく接続されていることを確認します。
- 各ポイントごとに適切な整定時間を置いてソース信号を安定化させてください。
- 試験装置と 260x 型は直接接続し、接続の途中にスキャナ等のスイッチング機器を挿入しないでください。
- 校正作業中にエラーが発生すると、260x 型は該当するエラーメッセージを発生します。詳しくは「エラーのまとめ」（p.B-2）を参照してください。

警告

コモンモード電圧（LO と筐体 GND 間の電圧）の許容最大値は 250 VDC です。これを超える電圧の印加は絶縁破壊を引き起こして感電事故発生の原因となります。

260x 型の端子は設置カテゴリ I の回路との接続に適合するように設計されています。したがって、260x 型の端子を CAT II、CAT III、CAT IV 回路には接続しないでください。ソースメータの端子を CAT I 規格を超える回路に接続すると装置の損傷を招き、オペレータを危険な電圧に曝す恐れがあります。

出力端子とガード端子には危険な高電圧が印加される可能性があります。感電事故防止のため、装置が通電状態のときには絶対に 260x 型の接続や接続切り離し作業を行わないでください。場合によっては致命的な傷害を引き起こす可能性があります。出力端子に接続されているケーブルを処理するときは、必ず事前に前面パネルから電源を切るか、または 260x 型背面の電源ケーブルを引抜いてください。装置をスタンバイ状態に入れたとしても、ハードウェアやソフトウェア等の障害によって出力が完全にオフになっていない可能性がありますのでご注意ください。

校正サイクル

少なくとも 1 年に 1 回は校正を行って、装置が規定の仕様から外れることのないようにしてください。

校正に使用できる推奨装置

校正での使用にお奨めできる試験装置をまとめて表 16-1 に示します。この表に示す仕様を満たすか、あるいはそれ以上の性能を持つ装置であれば、ここに列挙した以外の装置を使用してもかまいません。可能な限り、260x 型の仕様よりも 4 倍以上優れた確度を持つ試験装置を使用するべきです。

表 16-1
校正に使用できる推奨試験装置

機種	メーカー / モデル名	確度		
デジタルマルチメータ	Agilent 3458A	DC 電圧 [1]	90mV:	±8ppm
			0.9V:	±5ppm
			5.4V:	±8ppm
			36V:	±11ppm
		DC 電流 [1]	90nA:	±430ppm
			0.9µA:	±55ppm
			9µA:	±25ppm
			90µA:	±23ppm
			0.9mA:	±20ppm
			9mA:	±20ppm
			90mA:	±35ppm
			0.9A:	±110ppm
0.5Ω、250W、0.1% 精密抵抗	Isotek RUG-Z-R500-0.1-TK3	抵抗 [2]	0.5Ω:	±125ppm

1. 推奨モデルが指定の校正ポイント測定のために持つべき確度が 90 日仕様に記載されています。
2. 3A および 10A レンジでの校正に用いる抵抗は、実際の使用前にデジタルマルチメータの抵抗測定機能を使用して特性を正確に把握しておく必要があります。

校正エラー

オペレータの誤操作によって不適切な校正が行われる危険を防止するため、260x 型は各校正ステップごとにエラー発生の有無をチェックします。

エラー検出の有無は、リモートからステータスバイトの EAV（Error Available）ビット（ビット 2）を監視することによって知ることができます。（ステータスバイトを要求するにはクエリ *STB? を使用します。）エラーのタイプを装置に問い合わせるには errorqueue.next コマンドを使用してください。この問合せに応答して 260x 型はエラーの性質を知らせてくれます。エラーメッセージについては付録 B、ステータスバイト情報については付録 D をご覧ください。

校正

リモート校正を実施するときは、IEEE-488 バスまたは RS-232 ポートを介して以下の操作を行います。リモートコマンドと正しいパラメータ値についてはステップごとに項を改めて説明します。

校正ステップ

ステップシーケンス

校正ステップは表 16-2 に記載の順序にしたがって実施する必要があります。特に注記のある場合を除き、すべてのステップで 2 線式接続（ローカルセンシング）を用います。それぞれのレンジで 4 点校正を実施します。

- + ゼロ
- + フルスケール
- - ゼロ
- - フルスケール

パラメータ値

フルスケールパラメータには実際には表 16-2 に示す値の 90% を指定します。2 つのゼロパラメータには厳密な意味でのゼロを送れないことに注意してください。その代わりに非常に小さな値、たとえば 1e-60 または -1e-60 を送ります。

センスモード

校正ステップで使用するセンスモード一覧を表 16-2 に示します。ソースレンジと測定レンジはすべて LOCAL センスモードを使用して校正します。これに加えて、100mV のソースと測定レンジは REMOTE センスモードで、1mA と 1mV ソースレンジは CALA センスモードでも校正が可能です。

表 16-2
校正ステップ

機能	校正ステップ[2]	校正ポイント[4]	センスモード[5]
電圧ソースと測定[1]	100mV	±1e-60, ±90mV	smuX.SENSE_LOCAL
	100mV	±1e-60, ±90mV	smuX.SENSE_REMOTE
	1V	±1e-60, ±0.9V	smuX.SENSE_LOCAL
	1V	±1e-60, ±0.9V	smuX.SENSE_CALA
	6V	±1e-60, ±5.4V	smuX.SENSE_LOCAL
	40V	±1e-60, ±36V	smuX.SENSE_LOCAL
電流ソースと測定[1]	100nA	±1e-60, ±90nA	smuX.SENSE_LOCAL
	1µA	±1e-60, ±0.9µA	smuX.SENSE_LOCAL
	10µA	±1e-60, ±9µA	smuX.SENSE_LOCAL
	100µA	±1e-60, ±90µA	smuX.SENSE_LOCAL
	1mA	±1e-60, ±0.9mA	smuX.SENSE_LOCAL
	1mA	±1e-60, ±0.9mA	smuX.SENSE_CALA
	10mA	±1e-60, ±9mA	smuX.SENSE_LOCAL
	100mA	±1e-60, ±90mA	smuX.SENSE_LOCAL
	1A	±1e-60, ±0.9A	smuX.SENSE_LOCAL
	3A	±1e-60, ±2.4A	smuX.SENSE_LOCAL
	10A[3]	±1e-60, ±2.4A	smuX.SENSE_LOCAL

1. SENSE_CALA センスステップのソースのみを校正します。
2. 表に示された順に従ってステップを進めてください。
3. 10A はレンジ校正変更のために用意されているレンジであり、通常は使用しません。
4. ゼロ校正ポイントに実際の 0 は指定できません。その代わりに非常に小さな値、たとえば ± 1e-60 を送ってください。校正の極性も手順に示された順に設定してください。
5. CALA センスモードに切換える前に出力を OFF にしてください。

校正コマンド

リモート校正コマンドをまとめて表 16-3 に示します。これらのコマンドのさらに詳細な説明についてはセクション 12 をご覧ください。

表 16-3
校正コマンド

コマンド[1]	説明
smuX.cal.date = caldate	校正日付を設定
smuX.cal.due = caldue	次回の校正予定日付を設定
smuX.cal.lock()	校正ロックアウト
smuX.cal.password = "newpassword"	パスワードを "newpassword" に変更
smuX.cal.polarity = polarity	極性設定： smuX.CAL_AUTO (極性自動選択) smuX.CAL_NEGATIVE (負極性) smuX.CAL_POSITIVE (正極性)
smuX.cal.restore([calset])	校正用定数セットのローディング： smuX.CALSET_NOMINAL (公称定数) smuX.CALSET_FACTORY (工場定数) smuX.CALSET_DEFAULT (公称定数) smuX.CALSET_PREVIOUS (前回使用した定数)
smuX.cal.save()	定数を DEFAULT 校正セットとして不揮発性メモリに保存
calstate = smuX.cal.state	校正状態情報を要求： smuX.CALSTATE_CALIBRATING smuX.CALSTATE_LOCKED smuX.CALSTATE_UNLOCKED
smuX.cal.unlock("password")	校正のロックを解除（デフォルトパスワード：KI002601 または KI2602）
smuX.measure.calibratei(range, cp1measured, cp1reference, cp2measured, cp2reference)	電流測定レンジ校正：[2] ±range (校正の対象となる測定レンジ) cp1measured (260x 型による校正ポイント 1 測定値) cp1reference (校正ポイント 1 の基準測定値) cp2measured (260x 型による校正ポイント 2 測定値) cp2reference (校正ポイント 2 の基準測定値)
smuX.measure.calibratev(range, cp1measured, cp1reference, cp2measured, cp2reference)	電圧測定レンジ校正：[2] ±range (校正の対象となる測定レンジ) cp1measured (260x 型による校正ポイント 1 測定値) cp1reference (校正ポイント 1 の基準測定値) cp2measured (260x 型による校正ポイント 2 測定値) cp2reference (校正ポイント 2 の基準測定値)

1. 2601 型：smuX = smua、2602 型：smuX = smua (Channel A) または smub (Channel B)
2. 校正ポイント 1 はレンジの 0% 近傍で測定を行い、校正ポイント 2 はレンジの 90% 近傍で測定を行ってください。校正ポイントについては表 16-2 を参照。

表 16-3 (続き)
校正コマンド

コマンド[1]	説明
smuX.source.calibratei(range, cp1expected, cp1reference, cp2expected, cp2reference)	電流ソースレンジ校正 :[2] ±range (校正の対象となるレンジ) cp1expected (校正ポイント 1 用にプログラムされたソース値) cp1reference (校正ポイント 1 の基準測定値) cp2expected (校正ポイント 2 用にプログラムされたソース値) cp2reference (校正ポイント 2 の基準測定値)
smuX.source.calibratev(range, cp1expected, cp1reference, cp2expected, cp2reference)	電圧ソースレンジ校正 :[2] ±range (校正の対象となるレンジ) cp1expected (校正ポイント 1 用にプログラムされたソース値) cp1reference (校正ポイント 1 の基準測定値) cp2expected (校正ポイント 2 用にプログラムされたソース値) cp2reference (校正ポイント 2 の基準測定値)

1. 2601 型 : smuX = smua、2602 型 : smuX = smua (Channel A) または smub (Channel B)
2. 校正ポイント 1 はレンジの 0% 近傍で測定を行い、校正ポイント 2 はレンジの 90% 近傍で測定を行ってください。校正ポイントについては表 16-2 を参照。

校正手順

ステップ 1. 260x 型の校正準備

1. コントロールに使用する IEEE-488 インターフェイスまたは RS-232 ポートと 260x 型を接続します（シールド付インターフェイスケーブル使用）。
2. 260x 型と試験装置に電源を入れ、校正実行前に 2 時間以上のウォームアップを行います。
3. IEEE-488 または RS-232 インターフェイスのパラメータが正しく設定されていることを確認します（インターフェイスの設定には MENU キーと COMMUNICATION メニューを使用します）。

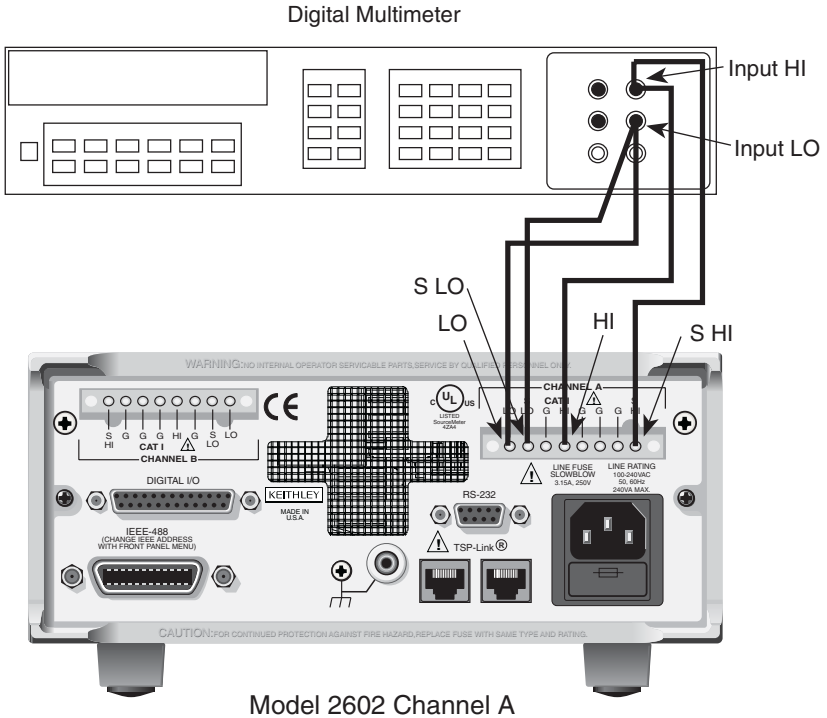
ステップ 2. 電圧校正

1. 260x 型とデジタルマルチメータを図 16-1 に示す 4 線式で接続し、マルチメータの DC 電圧機能を選択します。
2. 次のコマンドを送って電圧校正の初期化を行います。

```
smua.cal.unlock("KI002602")
smua.reset()
smua.source.func = smua.OUTPUT_DCVOLTS
```

（smua.cal.unlock のコマンドを実際の校正用パスワードで置き換えます。2601 型のデフォルトパスワードは "KI002601"、2602 型は "KI002602" です。）

図 16-1
電圧校正のための接続



3. 表 16-2（p.16-6）に示す各校正ステップを以下の手順にしたがって実行します。
   a. 次のコマンドを使用して校正の対象となるレンジを選択します。
      ```
      smua.source.rangev = range
      ```
      （校正に使用する測定レンジを指定する必要がないことに注意してください。）たとえば、1 V レンジを選択するのであれば次のコマンドを送ります。
      ```
      smua.source.rangev = 1
      ```
   b. 表 16-2 に記載の校正ステップにしたがって正しいセンスモードを選択します。たとえば：
      ```
      smua.sense = smua.SENSE_LOCAL
      ```
   c. 正極性校正を選択し、ソース出力を正のゼロに設定します。
      ```
      smua.cal.polarity = smua.CAL_POSITIVE
      smua.source.levelv = 1e-60
      ```
   d. 出力を ON にします。
      ```
      smua.source.output = smua.OUTPUT_ON
      ```

e. 指示値が安定するのを待ち、正のゼロ値におけるマルチメータと 260x 型両方の電圧読み取り値を取得します。（この校正ステップを CALA センスモードで行った場合は 260x 型での測定は不要です。）2 つの測定値は時間とともに極めて近い値を示すはずです。260x 型のデータ取得には次のコマンドを使用します。
```
Z_rdg = smua.measure.v()
```
f. 出力を OFF にします。
```
smua.source.output = smua.OUTPUT_OFF
```
g. ソース出力を、現在のレンジの正のフルスケール値に設定します。たとえば次のコマンドを使用します。
```
smua.source.levelv = 0.9
```
h. 出力を ON にします。
```
smua.source.output = smua.OUTPUT_ON
```
i. 指示値が安定するのを待ち、正のフルスケール出力値におけるマルチメータと 260x 型両方の電圧読み取り値を取得します。（この校正ステップを CALA センスモードで行った場合は 260x 型での測定は不要です。）2 つの測定値は時間とともに極めて近い値を示すはずです。260x 型からのデータ取り込みには次のコマンドを使用します。
```
FS_rdg = smua.measure.v()
```
j. 出力を OFF にします。
```
smua.source.output = smua.OUTPUT_OFF
```
k. 次のソース校正コマンドを送ります。使用するパラメータは正のゼロと正のフルスケール値に対するマルチメータ読み取り値、正のゼロと正のフルスケールに対する実際のソース出力値です。
```
smua.source.calibratev(range,src_Z,DMM_Z_rdg,
src_FS,DMM_FS_rdg)
```
ここに、`range` = 現在の校正レンジ
`src_Z` = 正のゼロにおける 260x 型のソース出力値
`DMM_Z_rdg` = 正のゼロにおける DMM 指示値
`src_FS` = 正のフルスケールにおける 260x 型のソース出力値
`DMM_FS_rdg` = 正のフルスケールにおける DMM 指示値
1V レンジであるとすれば、標準的なパラメータ値は次のようになります。
```
smua.source.calibratev(1,1e-60,1e-5,0.9,0.903)
```

l. このステップを CALA 以外のセンスモードで行った場合は次の測定校正コマンドを送ります。与えるパラメータはマルチメータと 260x 型の指示値、およびレンジ設定値です。
```
smua.measure.calibratev(range,Z_rdg,DMM_Z_rdg,
FS_rdg,DMM_FS_rdg)
```
ここに、range = 現在の校正レンジ
Z_rdg = 正のゼロにおける 260x 型測定値
DMM_Z_rdg = 正のゼロにおける DMM 測定値
FS_rdg = 正のフルスケールにおける 260x 型測定値
DMM_FS_rdg = 正のフルスケールにおける DMM 測定値
1V レンジでの標準的なパラメータ値は次のようになります。
```
smua.measure.calibratev(1,1e-4,1e-5,0.92,0.903)
```

m. 負極性校正を選択し、ソース出力を負のゼロに設定します。たとえば :
```
smua.cal.polarity = smua.CAL_NEGATIVE
smua.source.levelv = -1e-60
```

n. 出力を ON にします。
```
smua.source.output = smua.OUTPUT_ON
```

o. 指示値が安定するのを待ち、負のゼロ値におけるマルチメータと 260x 型両方の電圧読み取り値を取得します。（この校正ステップを CALA センスモードで行った場合は 260x 型での測定は不要です。）2 つの測定値は時間とともに極めて近い値を示すはずです。260x 型からのデータ取り込みには次のコマンドを使用します。
```
Z_rdg = smua.measure.v()
```

p. 出力を OFF にします。
```
smua.source.output = smua.OUTPUT_OFF
```

q. ソース出力を負のフルスケール値に設定します。たとえば :
```
smua.source.levelv = -0.9
```

r. 出力を ON にします。
```
smua.source.output = smua.OUTPUT_ON
```

s. 指示値が安定するのを待ち、負のフルスケール出力値におけるマルチメータと 260x 型両方の電圧読み取り値を取得します。（この校正ステップを CALA センスモードで行った場合は 260x 型での測定は不要です。）2 つの測定値は時間とともに極めて近い値を示すはずです。260x 型からのデータ取り込みには次のコマンドを使用します。
```
FS_rdg = smua.measure.v()
```

t. 出力を OFF にします。
```
smua.source.output = smua.OUTPUT_OFF
```

u. 次のソース校正コマンドを送ります。使用するパラメータは負のゼロと負のフルスケール値に対するマルチメータ指示値、負のゼロと負のフルスケールにおける実際のソース出力値です。

```
smua.source.calibratev(-range,src_Z,DMM_Z_rdg,
src_FS,DMM_FS_rdg)
```

ここに、-range = 現在の校正レンジ値の極性を変えた値
src_Z = 負のゼロにおける 260x 型のソース出力値
DMM_Z_rdg = 負のゼロにおける DMM 指示値
src_FS = 負のフルスケールにおける 260x 型のソース出力値
DMM_FS_rdg = 負のフルスケールにおける DMM 指示値

レンジが 1V であるとすれば、標準的なパラメータ値は次のようになります。

```
smua.source.calibratev(-1,-1e-60,-1e-4,-0.9,-0.896)
```

v. このステップを CALA 以外のセンスモードで行った場合は次の測定校正コマンドを送ります。与えるパラメータはマルチメータと 260x 型の指示値、およびレンジ設定値です。

```
smua.measure.calibratev(-range,Z_rdg,DMM_Z_rdg,
FS_rdg,DMM_FS_rdg)
```

ここに、-range = 現在の校正レンジ値の極性を変えた値
Z_rdg = 負のゼロにおける 260x 型測定値
DMM_Z_rdg = 負のゼロにおける DMM 測定値
FS_rdg = 負のフルスケールにおける 260x 型測定値
DMM_FS_rdg = 負のフルスケールにおける DMM 測定値

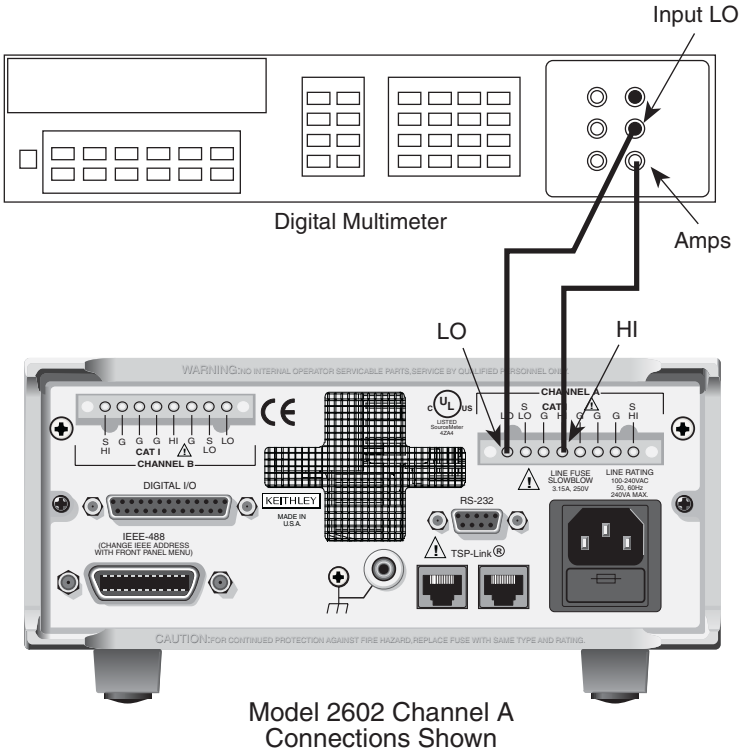
1V レンジでの標準的なパラメータ値は次のようになります。

```
smua.measure.calibratev(-1,-1e-4,-1e-6,-0.89,-0.896)
```

4. 表 16-2 に記載の 6 段階の電圧ステップすべてについて a から v までの操作を行ってください。以上の操作が完了してから電流校正へ移行します。
5. 自動極性モードを選択します。

```
smua.cal.polarity = smua.CAL_AUTO
```

図 16-2
電流校正のための接続（100nA ～ 1A レンジ）



Model 2602 Channel A
Connections Shown

ステップ 3. 電流校正

1. 260x 型をデジタルマルチメータに接続し（図 16-2 参照）、マルチメータの DC 電流機能を選択します。
2. 次のコマンドを送って電流校正の初期化を行います。
   `smua.source.func = smua.OUTPUT_DCAMPS`
3. 表 16-2（p.16-6）に示す各校正ステップを実行します。
   a. 次のコマンドを使用して校正の対象となるレンジを選択します。
      `smua.source.rangei = range`
      （校正に使用する測定レンジを指定する必要がないことに注意してください。）たとえば、1A レンジを選択するのであれば次のコマンドを送ります。
      `smua.source.rangei = 1`
   b. 表 16-2 に記載の校正ステップにしたがって正しいセンスモードを選択します。たとえば：
      `smua.sense = smua.SENSE_LOCAL`

c. 正極性校正を選択し、ソース出力を正のゼロに設定します。
```
smua.cal.polarity = smua.CAL_POSITIVE
smua.source.leveli = 1e-60
```
d. 出力を ON にします。
```
smua.source.output = smua.OUTPUT_ON
```
e. 指示値が安定するのを待ち、正のゼロ値におけるマルチメータと 260x 型両方の電流読み取り値を取得します。（この校正ステップを CALA センスモードで行った場合は 260x 型での測定は不要です。）2 つの測定値は時間とともに極めて近い値を示すはずです。260x 型からのデータ取り込みには次のコマンドを使用します。
```
Z_rdg = smua.measure.i()
```
f. 出力を OFF にします。
```
smua.source.output = smua.OUTPUT_OFF
```
g. ソース出力を、現在のレンジの正のフルスケール値に設定します。たとえば次のコマンドを使用します。
```
smua.source.leveli = 0.9
```
h. 出力を ON にします。
```
smua.source.output = smua.OUTPUT_ON
```
i. 指示値が安定するのを待ち、正のフルスケール出力値におけるマルチメータと 260x 型両方の電圧読み取り値を取得します。（校正を CALA センスモードで行った場合は 260x 型での測定は不要です。）2 つの測定値は時間とともに極めて近い値を示すはずです。260x 型からのデータ取り込みには次のコマンドを使用します。
```
FS_rdg = smua.measure.i()
```
j. 出力を OFF にします。
```
smua.source.output = smua.OUTPUT_OFF
```
k. 次のソース校正コマンドを送ります。使用するパラメータはゼロと正のフルスケールに対するマルチメータ読み取り値、およびゼロと正のフルスケールに対する実際のソース出力値です。
```
smua.source.calibratei(range,src_Z,DMM_Z_rdg,
src_FS,DMM_FS_rdg)
```
ここに、`range` = 現在の校正レンジ
`src_Z` = 正のゼロにおける 260x 型のソース出力値
`DMM_Z_rdg` = 正のゼロにおける DMM 指示値
`src_FS` = 正のフルスケールにおける 260x 型のソース出力値
`DMM_FS_rdg` = 正のフルスケールにおける DMM 指示値
レンジを 1A とすれば、標準的なパラメータ値は次のようになります。
```
smua.source.calibratei(1,1e-60,1e-4,0.9,0.88)
```

l. このステップを CALA 以外のセンスモードで行った場合は、次の測定校正コマンドを送ります。与えるパラメータはマルチメータと 260x 型の指示値、およびレンジ設定値です。
```
smua.measure.calibratei(range,Z_rdg,DMM_Z_rdg,
FS_rdg,DMM_FS_rdg)
```
ここに、`range` = 現在の校正レンジ
`Z_rdg` = 正のゼロにおける 260x 型測定値
`DMM_Z_rdg` = 正のゼロにおける DMM 測定値
`FS_rdg` = 正のフルスケールにおける 260x 型測定値
`DMM_FS_rdg` = 正のフルスケールにおける DMM 測定値
1A レンジでの標準的なパラメータ値は次のようになります。
```
smua.measure.calibratei(1,1e-5,1e-4,0.89,0.88)
```

m. 負極性校正を選択し、ソース出力を負のゼロに設定します。たとえば：
```
smua.cal.polarity = smua.CAL_NEGATIVE
smua.source.leveli = -1e-60
```

n. 出力を ON にします。
```
smua.source.output = smua.OUTPUT_ON
```

o. 指示値が安定するのを待ち、負のゼロ値におけるマルチメータと 260x 型両方の電流読み取り値を取得します。（この校正ステップを CALA センスモードで行った場合は 260x 型での測定は不要です。）2 つの測定値は時間とともに極めて近い値を示すはずです。260x 型からのデータ取り込みには次のコマンドを使用します。
```
Z_rdg = smua.measure.i()
```

p. 出力を OFF にします。
```
smua.source.output = smua.OUTPUT_OFF
```

q. ソース出力を負のフルスケール値に設定します。たとえば：
```
smua.source.leveli = -0.9
```

r. 出力を ON にします。
```
smua.source.output = smua.OUTPUT_ON
```

s. 指示値が安定するのを待ち、負のフルスケール出力値におけるマルチメータと 260x 型両方の電流読み取り値を取得します。（この校正ステップを CALA センスモードで行った場合は 260x 型での測定は不要です。）2 つの測定値は時間とともに極めて近い値を示すはずです。260x 型からのデータ取り込みには次のコマンドを使用します。
```
FS_rdg = smua.measure.v()
```

t. 出力を OFF にします。
```
smua.source.output = smua.OUTPUT_OFF
```

u. 次のソース校正コマンドを送ります。使用するパラメータは負のゼロと負のフルスケール値に対するマルチメータ指示値、および負のゼロと負のフルスケールにおける実際のソース出力値です。

```
smua.source.calibratei(-range,src_Z,DMM_Z_rdg,
src_FS,DMM_FS_rdg)
```

ここに、-range = 現在の校正レンジ値の極性を変えた値
src_Z = 負のゼロにおける 260x 型のソース出力値
DMM_Z_rdg = 負のゼロにおける DMM 指示値
src_FS = 負のフルスケールにおける 260x 型のソース出力値
DMM_FS_rdg = 負のフルスケールにおける DMM 指示値

レンジを 1A とすれば、標準的なパラメータ値は次のようになります。

```
smua.source.calibratei(-1,-1e-60,-1e-5,-0.9,-0.892)
```

v. このステップを CALA 以外のセンスモードで行った場合は次の測定校正コマンドを送ります。与えるパラメータはマルチメータと 260x 型の指示値、およびレンジ設定値です。

```
smua.measure.calibratei(-range,Z_rdg,DMM_Z_rdg,
FS_rdg,DMM_FS_rdg)
```

ここに、-range = 現在の校正レンジ値の極性を変えた値
Z_rdg = 負のゼロにおける 260x 型測定値
DMM_Z_rdg = 負のゼロにおける DMM 測定値
FS_rdg = 負のフルスケールにおける 260x 型測定値
DMM_FS_rdg = 負のフルスケールにおける DMM 測定値

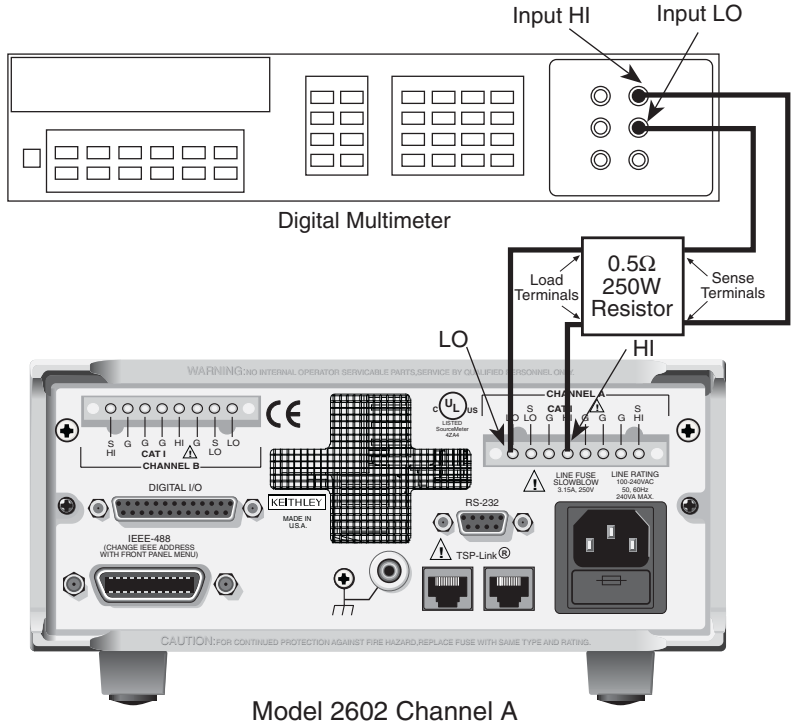
1A レンジでの標準的なパラメータ値は次のようになります。

```
smua.measure.calibratei(-1,-1e-4,-1e-5,-0.91,-0.892)
```

4. 100nA から 1A までの各レンジについて上記 a から v までの測定を行ってください。以上すべての測定が完了してから 3A と 10A レンジの校正へ移行します。
5. 接続を図 16-3 に示す状態へと変更します（図のように 4 線式接続を用いて 0.5Ω 抵抗を接続します）。
6. DMM の DC 電圧機能を選択します。
7. 3A と 10A レンジを使用して a から v までのステップを繰り返します。DMM の電圧指示値と予め厳密な値の分っている 0.5Ω 抵抗とから電流値を計算（I=V/R）します。
8. 自動極性モードを選択します。

```
smua.cal.polarity = smua.CAL_AUTO
```

図 16-3
電流校正のための接続（3A、10A レンジ）



ステップ 4. 校正日付のプログラム

次のコマンドを使用して校正日付および次回の校正予定日付を設定します。

```
smua.cal.date = os.time({year=2005, month=1, day=1})
smua.cal.due = os.time({year=2006, month=1, day=1})
```

設定したい日付の年、月、日を指定します（オプションとして時間、分も設定可能）。（同様に秒情報を含めることも可能ですが、内部の日付データ保存形式の精度上の制約のためこの情報は無視されます。）指定できる数値の範囲は、年が 2005 から 2037 まで、月が 1 から 12 まで、日が 1 から 31 までです。

ステップ 5. 校正定数の保存

ここまでの操作で校正は完了しました。次のコマンドを実行して校正定数を不揮発性メモリに保存します。

```
smua.cal.save()
```

この save コマンドを実行するまでは校正の結果は一時メモリに存在し、電源が切れると消えてしまいます。

ステップ 6. 校正をロックアウト

校正作業が完了したならば、不用意な校正の変更を禁止（ロックアウト）するために次のコマンドを実行してください。

```
smua.cal.lock()
```

ステップ 7. 2602 型チャンネル B の校正作業

2602 型をご使用の場合はチャンネル B についてもここまで説明したすべての校正作業を行う必要があります。その際、以下の点に注意してください。

- 試験用の接続にはチャンネル B 端子を使用する。
- すべてのコマンドの “smua” を “smub” に置き換える。

17
日常のメンテナンス

セクション 17 のトピックス

はじめに , 17-2

電源ヒューズの交換 , 17-2

前面パネルのテスト , 17-3
キー動作テスト , 17-3
表示パターンテスト , 17-4

はじめに

このセクションでは、オペレータに実施していただく日常メンテナンスの方法を説明します。

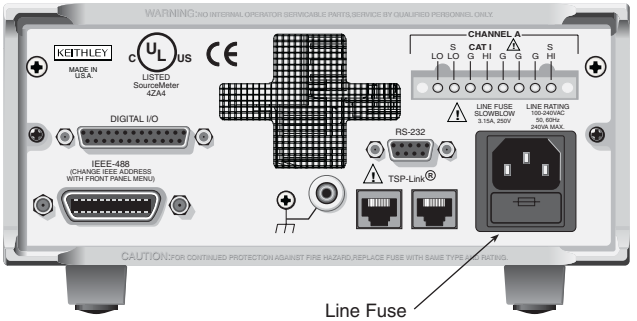
電源ヒューズの交換

警告 **電源ヒューズを交換するときは、作業着手の前に必ず電源コードを背面パネルから引抜き、装置に接続されている試験用の導線類もすべて取り外してください。**

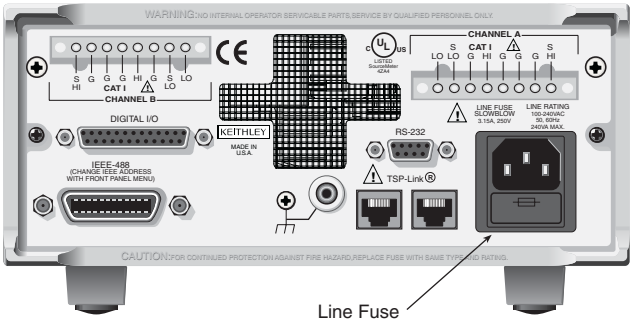
電源ヒューズは装置の背面パネルからアクセスします（AC 電源リセプタクルのすぐ下に配置されています：図 17-1 参照）。

図 17-1
電源ヒューズの交換

Model 2601



Model 2602



以下の手順にしたがって電源ヒューズを交換してください。

1. ヒューズキャリアをホルダに固定しているロックタブを慎重につかんで揺すり、ロックを外します。
2. ヒューズキャリアを引き出して、表 17-1 に指定された規格のヒューズと交換します。

注意　　装置の損傷防止のため、表 17-1 に指定されたタイプ以外のヒューズは絶対に使用しないでください。

3. ヒューズキャリアを元の位置に取付けます。

ヒューズの断線が何度も起るようであれば回路の異常が考えられますので修理が必要です。修理のため、装置を Keithley Instruments へ返送してください。

表 17-1
電源ヒューズ

電源電圧	定格	Keithley 部品番号
100-240V	250V、3.15A（スローブロー）、5 × 20mm	FU-106-3.15

前面パネルのテスト

前面パネルに関するテストには、前面パネルキーの機能チェックとディスプレイテストの 2 種類があります。

キー動作テスト

このテストは前面パネルのキーが正常に機能しているか否かをチェックします。以下の操作を行ってキーが正常に機能することを確認してください。

1. MENU キーを押して MAIN MENU を表示します。
2. TEST を選択してから ENTER または回転ノブを押すことにより、SELF- TEST MENU を表示させます。
3. DISPLAY-TESTS を選択し、続いて ENTER または回転ノブを押すと次のメニューが表示されます。
   FRONT PANEL TESTS
   KEYS DISPLAY_PATTERNS
4. ここで KEYS を選択して ENTER または回転ノブを押すことによりテストが始まります。パネル上のいずれかのキーを押したときに該当するラベル名が表示されれば、そのキーは正常に機能しています。また、キーから指を離すと“No keys pressed”と表示されます。
5. EXIT キーをチェックするときも同様に EXIT キーを押します。ただし、2 回連続して EXIT キーを押すとテストが中止され、SELF-TEST MENU へ戻ります。メニューから抜け出すときは EXIT キー何回か押してください。

表示パターンテスト

表示パターンテストを実施することにより、ディスプレイ（真空蛍光表示）の各ピクセルと表示ランプが正常に機能しているか否かをチェックします。以下のステップを順に実行してディスプレイをテストしてください。

1. MENU キーを押して MAIN MENU を表示します。
2. TEST を選択してから ENTER または回転ノブを押すことにより、SELF- TEST MENU を表示させます。
3. DISPLAY-TESTS を選択し、続いて ENTER または回転ノブを押すと次のメニューが表示されます。
   FRONT PANEL TESTS
   KEYS DISPLAY_PATTERNS
4. ここで DISPLAY_PATTERNS を選択して、ENTER または回転ノブを押すことによりディスプレイのテストが始まります。テストは次に説明する 3 種類の表示パターンを使用します。ENTER または回転ノブを押すたびに、次のテストシーケンスが選択されて表示パターンが切り換わります。
   - 格子模様パターンが表示され、通常操作で ON になる通知ランプのみが点灯。
   - 格子模様パターン（1 つおきにピクセルが ON）が表示され、全部の通知ランプが点灯。
   - 数字の桁（および、隣接する通知ランプ）が順番に点灯。選択された桁の全ピクセルが点灯します。
5. 表示テストを終了するには、EXT キーを押して FRONT PANEL TESTS MENU へ戻ってください。さらに EXIT キーを何回か押すとメニュー構造から抜け出します。

A
仕様

付録 A のトピックス

2601/ 2602 型システムソースメータ仕様

仕様条件

この資料には 2601 型および 2602 型の仕様および追加情報が記載されています。仕様データは 2601 / 2602 型を試験する際の対象となる基準値であり、2601 / 2602 型はこれらの仕様を満足する状態で工場から出荷されます。補足 / 典型値として示される値は使用温度 23 ℃ における有用な参考値として提供されるものであり、その達成が保証されるものではありません。ソースおよび測定の確度は、ソースメータ CHANNEL A（2601 / 2602 型）またはソースメータ CHANNEL B（2602 型）を以下の項目を満たす条件で使用した場合の端子における値を規定するものです。

1. 23 ℃ +5 ℃、相対湿度 <70%
2. 2 時間のウォームアップ終了後
3. 標準速度（1 NPLC）
4. A/D 自動ゼロ機能 ON
5. リモートセンス動作、または適正にゼロ補正されたローカル動作
6. 校正周期：1 年

ソース仕様

電圧プログラミングの確度 [1]

レンジ	プログラミング分解能	確度（1 年）23 ℃ ± 5 ℃ ± (% rdg. + volts)	ノイズ (peak-peak) 0.1 ～ 10Hz
100.000mV	1 μ V	0.02% + 250 μ V	20 μ V
1.00000V	10 μ V	0.02% + 400 μ V	50 μ V
6.00000V	10 μ V	0.02% + 1.8mV	100 μ V
40.0000V	100 μ V	0.02% + 12mV	500 μ V

温度係数（0°～ 18 ℃ & 28°～ 50 ℃）：± (確度仕様値× 0.15)/ ℃
最大出力パワー、ソース / シンクリミット [2]：1 チャンネルあたり最大 40.4W。4 象限ソース / シンク動作：± 40.4V @ ± 1.0A、± 6.06V @ ± 3.0A
電圧制御：**電源ライン**：レンジの 0.01% **負荷**：レンジの 0.01%+ 100 μ V
ノイズ、10Hz ～ 20MHz（peak-peak）：抵抗性負荷に対して 25mV（典型値）
電流リミット / コンプライアンス [3]：双方の極性の電流リミット（コンプライアンス）を単一値で設定。最小値は 10nA。確度は電流ソースに同じ。
オーバーシュート：<0.1% + 10mV（典型値）（ステップサイズ：レンジの 10 ～ 90%、抵抗性負荷、最大電流リミット / コンプライアンス）
ガードオフセット電圧：<10mV（典型値）（Iout <= 100mA）

電流プログラミングの確度

レンジ	プログラミング分解能	確度（1 年）23 ℃ ± 5 ℃ ± (% rdg. + amps)	ノイズ (peak-peak) 0.1 ～ 10Hz
100.000nA	1pA	0.06 + 100pA	5pA
1.00000 μ A	10pA	0.03% + 600pA	15pA
10.0000 μ A	100pA	0.03% + 2nA	50pA
100.000 μ A	1nA	0.03% + 30nA	2nA
1.00000mA	10nA	0.03% + 200nA	5nA
10.0000mA	100nA	0.03% + 3 μ A	200nA
100.000mA	1 μ A	0.03% + 20 μ A	500nA
1.00000A	10 μ A	0.05% + 900 μ A	50 μ A
3.00000A [2]	10 μ A	0.06% + 1.5mA	150 μ A

温度係数（0°～ 18 ℃ & 28°～ 50 ℃）：± (確度仕様値× 0.15)/ ℃
最大出力パワー、ソース / シンクリミット [2]：1 チャンネルあたり最大 40.4W。4 象限ソース / シンク動作：± 1.01A @ ± 40.0V、± 3.03A @ ± 6.0V
電流制御：電源ライン：レンジの 0.01%。負荷：レンジの 0.01%+ 100pA
電圧リミット / コンプライアンス [4]：双方の極性の電圧リミット（コンプライアンス）を単一値で設定。最小値は 10mV。確度は電圧ソースに同じ。
オーバーシュート：<0.1%（典型値）（ステップサイズ：レンジの 10 ～ 90%、抵抗性負荷、試験の追加条件については「電流ソース出力整定時間」の項を参照）

追加ソース仕様

過渡応答時間：10% ～ 90% の負荷変動に対応して出力が 0.1% までに回復する時間：<70 μ s
電圧ソース出力整定時間：固定レンジでソースレベルコマンドが処理されたときに、最終値の 0.1% 以内に到達するのに要する時間。
- 100mV、1V レンジ：<50 μ s（典型値）
- 6V レンジ：<100 μ s（典型値）
- 40V レンジ：<150 μ s（典型値）

電流ソース出力整定時間：固定レンジでソースレベルコマンドが処理されたときに、最終値の 0.1% 以内に到達するのに要する時間。注記がある場合を除き、以下の値は Iout Rload = 2V の場合を示します。
- 3A ～ 10mA レンジ：<80 μ s（典型値）（電流 2.5A 未満、負荷 1.6 Ω 以上）
- 1mA レンジ：<100 μ s（典型値）
- 100 μ A レンジ：<150 μ s（典型値）
- 10 μ A レンジ：<500 μ s（典型値）
- 1 μ A レンジ：<2ms（典型値）
- 100nA レンジ：<20ms（典型値）

DC フローティング電圧：出力を筐体 GND から ± 250VDC の範囲でフローティング状態にすることが可能です。
リモートセンス動作レンジ [1]：
HI ～ SENSE HI 間最大電圧 =3V
LO ～ SENSE LO 間最大電圧 =3V
電圧出力余裕度：
40V レンジ：最大出力電圧 =42V - ソースリード線による全電圧降下（ソースリード線抵抗 1 Ω あたりの最大値）。
6V レンジ：最大出力電圧 =8V - ソースリード線による全電圧降下。
過温度保護：過大な熱負荷を内部で検出して装置を自動的にスタンバイモードへ移行。
電圧ソースレンジ切換時のオーバーシュート：100k Ω 負荷へのオーバーシュート、20MHz BW、300mV（典型値）。
電流ソースレンジ切換時のオーバーシュート：より大きなレンジの <5% + 300mV/ 抵抗負荷（典型値）（補足試験条件については「電流ソース出力整定時間」の項を参照。）

注

1. HI 導線による電圧降下 1V あたり、ソース確度仕様に 50 μ V を追加してください。
2. 負荷の大きさによらず、周囲温度 30 ℃以下でのフルパワーソース動作。周囲温度が 30 ℃ を超える場合、またはシンク動作におけるパワーディレーティングについては『Series 2600 Reference Manual』セクション 8 の動作限界の説明を参照。
3. シンクモード動作（象限 II,IV）では、対応する電流リミット確度仕様の値にリミットレンジとして 12%、リミット設定には ± 0.02% を追加。1A レンジの場合は確度誤差として 40mA を追加。
4. シンクモード動作（象限 II,IV）では、対応する電圧ソース仕様のコンプライアンスレンジに 10%、およびリミット設定に ± 0.02% を追加。100mV レンジの場合は確度誤差として 60mV を追加。

仕様は改良のために予告なく変更されることがあります。

2601/ 2602 型システムソースメータ仕様

計器仕様

電圧測定確度[1]

レンジ	表示分解能[3]	入力抵抗	確度（1 年） 23 ℃ ± 5 ℃ ± (% rdg. + volts)
100.000mV	1 μ V	>10G Ω	0.015% + 150 μ V
1.00000V	10 μ V	>10G Ω	0.015% + 200 μ V
6.00000V	10 μ V	>10G Ω	0.015% + 1mV
40.0000V	100 μ V	>10G Ω	0.015% + 8mV

温度係数（0°～ 18 ℃ & 28°～ 50 ℃）：± (確度度仕様値× 0.15)/ ℃

電流測定確度

レンジ	表示分解能[3]	電圧負荷	確度（1 年） 23 ℃ ± 5 ℃ ± (% rdg. + amps)
100.000nA	1pA	<1mV	0.05 + 100pA
1.00000 μ A	10pA	<1mV	0.025% + 300pA
10.0000 μ A	100pA	<1mV	0.025% + 600pA
100.000 μ A	1nA	<1mV	0.02% + 12nA
1.00000mA	10nA	<1mV	0.02% + 60nA
10.0000mA	100nA	<1mV	0.02% + 1.2 μ A
100.000mA	1 μ A	<1mV	0.02% + 6 μ A
1.00000A	10 μ A	<1mV	0.03% + 700 μ A
3.00000A[2]	10 μ A	<1mV	0.05% + 1mA

温度係数（0°～ 18 ℃ & 28°～ 50 ℃）：± (確度仕様値× 0.15)/ ℃

補足計器仕様

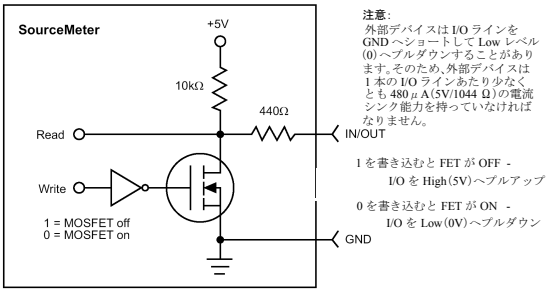
負荷インピーダンス：10,000pF まで安定（典型値）
コモンモード電圧：250VDC
コモンモード絶縁：>1G Ω、<4500pF
オーバーレンジ：ソースレンジの 101%、測定レンジの 102%
センスリード線の最大抵抗値：定格確度を満たすために 1k Ω 以内としてください。
センス入力インピーダンス：>10G Ω

注

1. HI 導線による電圧降下 1V あたり、ソース確度仕様値に 50 μ V を追加してください。
2. 4 線式リモートセンスのみ。
3. シングルチャンネル表示モードに適用されます。

一般事項

ホストインターフェイス：コンピュータコントロールインターフェイス。
IEEE-488:IEEE-488.1 準拠。IEEE-488.2 の共通コマンドおよびステータスモデルトポロジをサポート。
RS-232: ボーレート：300 ～ 115200 bps。データビット、パリティ、およびフローコントロール（RTS/CTS ハードウェアまたは不使用）はプログラム可能。アクティブなホストインターフェイスとしてプログラムしない場合、ソースメータは RS-232 インターフェイスを使用して他の装置をコントロールできます。
拡張インターフェイス：TSP-Link 拡張インターフェイスを使用すれば TSP 対応装置間相互での通信とトリガ送信が可能となります。
ケーブルタイプ：カテゴリ 5e 以上の LAN クロスオーバーケーブル。
長さ :TSP 対応装置間で最長 3m。
デジタル I/O インターフェイス：
コネクタ：25 ピン D タイプ（メス）
入力 / 出力ピン：I/O ビット× 14（下図参照）。



インターロックピン：アクティブ High 入力。インターロック入力がアクティブになると、ソースメータの各出力はインターロックのレベルが 1.5V 未満になると動作しなくなります。
5V 電源ピン：最大電流 600mA、ソリッドステートヒューズで保護されています。
供給電源：100 ～ 240VAC、50 ～ 60Hz（手動設定）、最大 240VA。
冷却機能：強制空冷。側面吸気・背面排気。ラック取付け時には少なくとも片側の側面からの吸気を阻害しないでください。
保証期間：1 年。
EMC: 欧州共同体指令 89/336/EEC、EN61326-1 準拠
安全性：欧州共同体指令 73/23/EEC、EN61010-1、および UL 61010-1 準拠
外形寸法：高さ 89mm ×幅 213mm ×奥行き 460mm（3 1/2 × 8 3/8 × 17 1/2 インチ）
卓上設置（ハンドル、足取付け）：高さ 104mm ×幅 238mm ×奥行き 460mm（4 1/8 × 9 3/8 × 17 1/2 インチ）
重量：2601 型：4.75kg (10.40 lbs)、2602 型：5.50kg (12.00 lbs)。
設置環境：室内使用専用。
標高：海抜 2000m 以下。
動作温度：0 ～ 50 ℃、35 ℃ まで 70%R.H。35 ℃ 以上 50 ℃ までは 3% R.H./ ℃ のディレーティングが必要。
保管温度：-25 ℃～ 65 ℃。

付属アクセサリ：

ケーブル・コネクタ：ソースメータ各チャンネル用 DUT インターフェイスコネクタキット。キットに含まれるフード付き端子コネクタはソースメータ側の測定端子に適合するように設計されています。TSP-Link ケーブル、電源ケーブル。
ドキュメント（印刷物）：ユーザマニュアル
電子媒体：下記の内容を含む CD-ROM：
- ユーザマニュアル、リファレンスマニュアル（PDF ファイル）
- Test Script Builder スクリプト開発用ソフトウェア。
- IVI/VISA ドライバ（VB、VC/C++、LabVIEW、LabWindows/CVI に対応）。

2601/ 2602 型システムソースメータ仕様

速度仕様[1]

最大スイープ動作速度（1 秒あたりの動作回数）、60Hz（50Hz）:

A/D 変換器速度	トリガ発生源	測定後メモリへ転送	測定後 GPIB へ転送	ソース測定後メモリへ転送	ソース測定後 GPIB へ転送	ソース測定合否判定をメモリへ転送	ソース測定合否判定を GPIB へ転送
0.001 NPLC	内部	10000 (10000)	8000 (8000)	5500 (5500)	3600 (3600)	4900 (4900)	3100 (3100)
0.001 NPLC	デジタル I/O	2700 (2650)	2100 (2100)	2300 (2300)	1900 (1875)	2200 (2150)	1800 (1775)
0.01 NPLC	内部	4000 (3500)	3600 (3200)	2750 (2700)	2300 (2100)	2800 (2500)	2100 (1975)
0.01 NPLC	デジタル I/O	1900 (1775)	1600 (1500)	1700 (1600)	1450 (1400)	1600 (1500)	1400 (1325)
0.1 NPLC	内部	565 (475)	555 (470)	540 (450)	510 (440)	535 (455)	505 (430)
0.1 NPLC	デジタル I/O	490 (420)	470 (405)	470 (410)	450 (390)	470 (400)	450 (390)
1.0 NPLC	内部	59 (49)	59 (49)	58 (49)	58 (48)	58 (49)	58 (48)
1.0 NPLC	デジタル I/O	58 (48)	58 (48)	58 (48)	57 (48)	57 (48)	57 (48)

シングル測定最高速度（1 秒あたりの動作回数）、60Hz（50Hz）:

A/D 変換器速度	トリガ発生源	測定後 GPIB へ転送	ソース測定後 GPIB へ転送	ソース測定合否判定を GPIB へ転送
0.001 NPLC	内部	1100 (1000)	880 (880)	840 (840)
0.01 NPLC	内部	950 (900)	780 (760)	730 (710)
0.1 NPLC	内部	390 (345)	355 (310)	340 (305)
1.0 NPLC	内部	57 (48)	56 (47)	56 (47)

測定レンジ切換えの最高速度：>4500/ 秒（典型値）
ソースレンジ切換えの最高速度：>1000/ 秒（典型値）
ソース機能切換えの最高速度：>500/ 秒（典型値）
外部トリガ入力：デジタル I/O インターフェイス信号をトリガ信号として使用できるように設定できます。
　入力潜在時間（トリガ入力から測定 / ソース切換え開始までの時間）：<150 μs（典型値）
　入力ジター：<100 μs（典型値）
コマンド処理時間：smux.source.levelv または smux.source.leveli コマンドを受け取ってから、出力が変化を始めるまでに要する最長時間。
　<1ms（典型値）

注

1. 試験条件について更に詳しくは『2600 シリーズリファレンスマニュアル』の速度仕様試験を説明した付録をご覧ください。

2601/ 2602 型システムソースメータ仕様

補足情報

前面パネルインターフェイス：2 行の真空蛍光ディスプレイ（VFD）およびキーパッドと回転ノブ。

表示内容：
- エラーメッセージとユーザ定義メッセージ
- ソースとリミット設定
- 電流および電圧測定値
- 不揮発性読み込みバッファに保存された測定値

キーパッド操作：
- ホストインターフェイス設定の変更
- 装置セットアップの保存と呼び戻し
- ファクトリ / ユーザ定義試験スクリプト（シーケンス）のローディングと実行（入力プロンプトの表示、結果の表示）。
- 測定値を不揮発性読み込みバッファへ保存

プログラミング：任意のホストインターフェイスからアクセス可能な組込み TSP（Test Script Processor）。個々の装置コントロールコマンドに応答します。装置コントロールコマンドと TSL 文（Test Script Language：分岐やループ処理、数値演算を記述可能）から構成された高速試験スクリプトに応答します。ホストからは独立して、メモリに保存されている高速試験スクリプトを実行します。

最小メモリ：3 メガバイト（TSL コード約 50,000 行に相当）

TSB（Test Script Builder）：TSP スクリプトを作成 / 実行 / 管理するための統合開発環境。他の TSP 対応装置との対話的な通信を可能にする装置コンソールを含みます。

要件：
- VISA（CD に NI-VISA が含まれています）
- Microsoft .NET Framework（CD に含まれます）
- Keithley I/O Layer（CD に含まれます）
- Pentium III 800MHz 以上の PC
- Microsoft Windows 98/NT/2000、または XP

ドライバ：IVI/VISA ドライバ（VB、VC/C++、LabVIEW、TestPoint、LabWindows/CVI に対応）。

読み込みバッファ：測定データの保存用に確保された不揮発性メモリ領域。読み込みバッファは測定値の要素（複数）を書き込む配列であり、各要素は次のような項目から構成されます。
- 測定値
- 測定ステータス
- タイムスタンプ
- ソース設定（測定値取得時の設定）
- レンジ情報

ソースメータのチャンネルごとに 2 つの読み込みバッファが予約されています。読み込みバッファへの書き込み / 読み出しには前面パネル（STORE/RECALL キー）、またはホストインターフェイスを使用します。

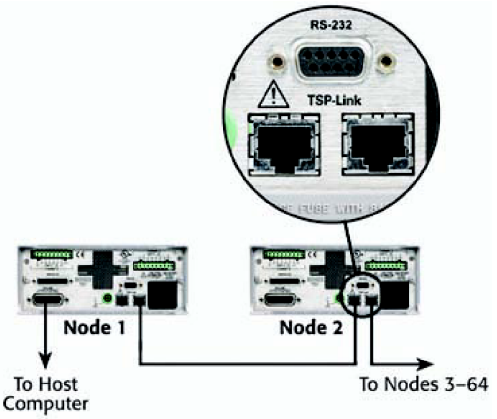
バッファ容量（タイムスタンプ、ソース設定保存）：>50,000 サンプル

バッファ容量（タイムスタンプ、ソース設定非保存）：>100,000 サンプル

バッテリバックアップ：リチウムイオン電池によるバックアップ。不揮発性メモリを 30 日間（23 ℃）保持、最短充電時間 4 時間。電池寿命：3 年（23 ℃）、1.5 年（50 ℃）。

ファクトリ TSP スクリプト：Keithley がサポートするアプリケーション専用スクリプトの詳細を www.keithley.com でご覧ください。

システム拡張：TSP-Link 拡張インターフェイスを使用すれば TSP 対応装置間相互での通信とトリガ送信が可能となります（下図参照）。



- ソースメータは複数の装置を連鎖接続するために、2 個の TSP-Link コネクタを備えています。
- TSP-Link で相互接続しておくと、コンピュータまたは個々のソースメータからすべての接続されたソースメータのリソースに自由にアクセスすることができます。
- TSP-Link は最高 64 個のノードを接続可能（ソースメータ 1 台が 1 つのノードになります）。

タイマ：1 MHz 入力で動作する 47 ビットのフリーランニングカウンタ。装置起動時にリセットされ、連続動作で 4 年間をカバーします。

タイムスタンプ：測定トリガ発生ごとに時刻（TIMER）を自動保存します。

分解能：1 μs

確度：50ppm

B
エラーメッセージとステータスメッセージ

付録 B のトピックス

はじめに , B-2

エラーのまとめ , B-2

エラーの読み取り , B-2

はじめに

この付録ではエラーレベルとエラーの読み方を説明し、後半にエラーメッセージを網羅した一覧表を掲載しています。

エラーのまとめ

エラー一覧を次ページから始まる表 B-2 に示します。表に記載のエラーレベルは次の意味を持ちます。

- NO_SEVERITY: 軽度のエラー
- INFORMATIONAL: 状況をユーザへ通知するためのメッセージ
- RECOVERABLE: それほど重大ではない、修復可能なエラー
- SERIOUS: 重大なエラーであるが、適切な是正措置により装置の操作継続は可能である
- FATAL: 装置は動作不能

エラーの読み取り

エラーが発生すると、エラーキューにエラーメッセージが書き込まれます(「キュー」、p.D-29、参照)。エラーキューの操作に関連するコマンドをまとめて表 B-1 に示します（詳しくはセクション 12 参照)。たとえば、下のコマンドはエラーキューに書かれている次のエラーの完全な情報を要求し、エラーのメッセージ部分を返します。

```
errorcode, message, severity, node = errorqueue.next()
print(message)
```

表 B-1
エラーキュー コマンド

エラーキュー コマンド	説明
errorqueue.clear()	エラーキュー内すべてのエラーをクリアします。
errorqueue.count	エラー / イベントキュー内に存在するメッセージの数を返します。
errorqueue.next()	エラーメッセージを要求します。

表 B-2
エラーー一覧

エラー番号	エラーレベル	エラーメッセージ
-440	RECOVERABLE	Query Unterminated After Indefinite Response
-430	RECOVERABLE	Query Deadlocked
-420	RECOVERABLE	Query Unterminated
-410	RECOVERABLE	Query Interrupted
-363	RECOVERABLE	Input Buffer Over-run
-362	RECOVERABLE	Framing
-361	RECOVERABLE	Parity
-360	RECOVERABLE	Communications
-350	RECOVERABLE	Queue Overflow
-330	RECOVERABLE	Self Test Failed
-315	RECOVERABLE	Configuration Memory Lost
-314	RECOVERABLE	Save/ Recall Memory Lost
-292	RECOVERABLE	Referenced name does not exist
-286	RECOVERABLE	TSP Syntax error, TSP Runtime error, TSP Double error, TSP File error, or TSP Unrecognized error
-285	RECOVERABLE	Program Syntax
-284	RECOVERABLE	Program Currently Running
-282	RECOVERABLE	Illegal Program Name
-281	RECOVERABLE	Cannot Create Program
-260	RECOVERABLE	Expression
-241	RECOVERABLE	Hardware Missing
-230	RECOVERABLE	Data Corrupt or Stale
-225	RECOVERABLE	Out of Memory or TSP Memory allocation error
-224	RECOVERABLE	Illegal Parameter Value
-223	RECOVERABLE	Too Much Data
-222	RECOVERABLE	Parameter Data Out of Range
-221	RECOVERABLE	Settings Conflict
-220	RECOVERABLE	Parameter
-215	RECOVERABLE	Arm Deadlock
-214	RECOVERABLE	Trigger Deadlock
-213	RECOVERABLE	Init Ignored
-212	RECOVERABLE	Arm Ignored

表 B-2 （続き）
エラーー覧

エラー番号	エラーレベル	エラーメッセージ
-211	RECOVERABLE	Trigger Ignored
-210	RECOVERABLE	Trigger
-203	RECOVERABLE	Command protected
-202	RECOVERABLE	Settings Lost Due to RTL
-201	RECOVERABLE	Invalid While in Local
-200	RECOVERABLE	Execution
-178	RECOVERABLE	Expression Data Not Allowed
-171	RECOVERABLE	Invalid Expression
-170	RECOVERABLE	Expression
-168	RECOVERABLE	Block Data Not Allowed
-161	RECOVERABLE	Invalid Block Data
-160	RECOVERABLE	Block Data
-158	RECOVERABLE	String Data Not Allowed
-154	RECOVERABLE	String Too Long
-151	RECOVERABLE	Invalid String Data
-150	RECOVERABLE	String Data
-148	RECOVERABLE	Character Data Not Allowed
-144	RECOVERABLE	Character Data Too Long
-141	RECOVERABLE	Invalid Character Data
-140	RECOVERABLE	Character Data
-128	RECOVERABLE	Numeric Data Not Allowed
-124	RECOVERABLE	Too Many Digits
-123	RECOVERABLE	Exponent Too Large
-121	RECOVERABLE	Invalid Character In Number
-120	RECOVERABLE	Numeric Data
-114	RECOVERABLE	Header Suffix Out Of Range
-113	RECOVERABLE	Undefined Header
-112	RECOVERABLE	Program Mnemonic Too Long
-111	RECOVERABLE	Header Separator
-110	RECOVERABLE	Command Header
-109	RECOVERABLE	Missing Parameter

表 B-2 （続き）
エラーー覧

エラー番号	エラーレベル	エラーメッセージ
-108	RECOVERABLE	Parameter Not Allowed
-105	RECOVERABLE	Trigger Not Allowed
-104	RECOVERABLE	Data Type
-103	RECOVERABLE	Invalid Separator
-102	RECOVERABLE	Syntax
-101	RECOVERABLE	Invalid Character
-100	RECOVERABLE	Command
0	INFORMATIONAL	No Error
100	INFORMATIONAL	Limit 1 Failed
101	INFORMATIONAL	Low Limit 2 Failed
102	INFORMATIONAL	Low Limit 2 Failed
103	INFORMATIONAL	Low Limit 3 Failed
104	INFORMATIONAL	High Limit 3 Failed
105	INFORMATIONAL	Active Limit Tests
106	INFORMATIONAL	Reading Available
107	INFORMATIONAL	Reading Overflow
108	INFORMATIONAL	Buffer Available
109	INFORMATIONAL	Buffer Full
110	INFORMATIONAL	Limit 4 Failed
111	INFORMATIONAL	Output Interlock Asserted
112	INFORMATIONAL	Temperature Limit Exceeded
113	INFORMATIONAL	Voltage Limit Exceeded
114	INFORMATIONAL	Source In Compliance
200	INFORMATIONAL	Operation Complete
300	INFORMATIONAL	Device Calibration
303	INFORMATIONAL	Device Sweeping
305	INFORMATIONAL	Waiting in Trigger Layer
306	INFORMATIONAL	Waiting in Arm Layer
310	INFORMATIONAL	Entering Idle Layer
408	INFORMATIONAL	Question Calibration
414	INFORMATIONAL	Command Warning

表 B-2 （続き）
エラー一覧

エラー番号	エラーレベル	エラーメッセージ
500	RECOVERABLE	Date of Calibration Not Set
501	RECOVERABLE	Next Date of Calibration Not Set
502	RECOVERABLE	Calibration Data Invalid
503	RECOVERABLE	Calibration Overflow
504	RECOVERABLE	Dac Calibration Underflow
505	RECOVERABLE	Source Offset Data Invalid
506	RECOVERABLE	Source Gain Data Invalid
507	RECOVERABLE	Measurement Offset Data Invalid
508	RECOVERABLE	Measurement Gain Data Invalid
509	RECOVERABLE	Not Permitted With Cal Locked
510	RECOVERABLE	Not Permitted With Cal Unlocked
601	RECOVERABLE	Reading Buffer Data Lost
602	RECOVERABLE	GPIB Address Lost
603	RECOVERABLE	Power On State Lost
604	RECOVERABLE	DC Calibration Data Lost
605	RECOVERABLE	Calibration Dates Lost
606	RECOVERABLE	GPIB Communication Language Lost
700	RECOVERABLE	Invalid System Communication
701	RECOVERABLE	ASCII Only With RS232
702	FATAL	Unresponsive digital FPGA
800	RECOVERABLE	Illegal With Storage Active
801	RECOVERABLE	Insufficient Vector Data
802	RECOVERABLE	Output Blocked By Interlock
803	RECOVERABLE	Not Permitted With Output Off
804	RECOVERABLE	Expression List Full
805	RECOVERABLE	Undefined Expression Exists
806	RECOVERABLE	Expression Not Found
807	RECOVERABLE	Definition Not Allowed
808	RECOVERABLE	Expression Cannot Be Deleted
809	RECOVERABLE	Source Memory Location Revised
810	RECOVERABLE	Output Blocked By Over Temperature

表 B-2 （続き）
エラーー覧

エラー番号	エラーレベル	エラーメッセージ
811	RECOVERABLE	Not An Operator Or Number
812	RECOVERABLE	Mismatched Parentheses
813	RECOVERABLE	Not a Number of Data Handle
814	RECOVERABLE	Mismatched Brackets
815	RECOVERABLE	Too Many Parentheses
816	RECOVERABLE	Entire Expression Not Parsed
817	RECOVERABLE	Unknown Token
818	RECOVERABLE	Parsing Mantissa
819	RECOVERABLE	Parsing Exponent
820	RECOVERABLE	Parsing Value
821	RECOVERABLE	Invalid Data Handle Index
822	RECOVERABLE	Too Small For Sense Range
823	RECOVERABLE	Invalid With Source Read Back On
824	RECOVERABLE	Cannot Exceed Compliance Range
825	RECOVERABLE	Invalid With Auto Ohms On
826	RECOVERABLE	Attempt to Exceed Power Limit
827	RECOVERABLE	Invalid With Ohms Guard On
828	RECOVERABLE	Invalid On 1 Amp Range
829	RECOVERABLE	Invalid On 1KV Range
830	RECOVERABLE	Invalid With Infinite Arm Count
831	RECOVERABLE	Invalid In Pulse Mode
900	FATAL	Internal System
1100	RECOVERABLE	Command Unavailable
1101	RECOVERABLE	Parameter Too Big
1102	RECOVERABLE	Parameter Too Small
1103	RECOVERABLE	Max Greater Than Min
1104	RECOVERABLE	Too many digits for param type
1105	RECOVERABLE	Too Many Parameters
1106	RECOVERABLE	Battery Not Present
1107	RECOVERABLE	Cannot modify factory menu
1108	RECOVERABLE	Menu name does not exist

表 B-2 （続き）
エラーー覧

エラー番号	エラーレベル	エラーメッセージ
1109	RECOVERABLE	Menu name already exists
1200	RECOVERABLE	TSPlink initialization failed
1201	RECOVERABLE	TSPlink initialization failed
1202	RECOVERABLE	TSPlink initialization failed
1203	RECOVERABLE	TSPlink initialization failed (possible loop in node chain)
1204	RECOVERABLE	TSPlink initialization failed
1205	RECOVERABLE	TSPlink initialization failed (no remote nodes found)
1206	RECOVERABLE	TSPlink initialization failed
1207	RECOVERABLE	TSPlink initialization failed
1208	RECOVERABLE	TSPlink initialization failed
1209	RECOVERABLE	TSPlink initialization failed
1210	RECOVERABLE	TSPlink initialization failed (node ID conflict)
1211	RECOVERABLE	Node %u is inaccessible
1212	RECOVERABLE	Invalid node ID
1400	RECOVERABLE	Expected at least %d parameters
1401	RECOVERABLE	Parameter %d is invalid
1402	RECOVERABLE	User scripts lost
1403	RECOVERABLE	Factory scripts lost
1404	RECOVERABLE	Invalid byte order
1405	RECOVERABLE	Invalid ASCII precision
1406	RECOVERABLE	Invalid data format
1500	RECOVERABLE	Invalid baud rate setting
1501	RECOVERABLE	Invalid parity setting
1502	RECOVERABLE	Invalid terminator setting
1503	RECOVERABLE	Invalid bits setting
1504	RECOVERABLE	Invalid flow control setting
1505	RECOVERABLE	Unknown error
1600	RECOVERABLE	Maximum GPIB message length exceeded
1800	RECOVERABLE	Invalid Digital Trigger Mode
1801	RECOVERABLE	Invalid Digital I/O Line
1802	RECOVERABLE	Digital Bit in Parameter Write Protected

表 B-2 （続き）
エラー一覧

エラー番号	エラーレベル	エラーメッセージ
2000	RECOVERABLE	Flash download error
2001	RECOVERABLE	Cannot flash with error in queue
4900	RECOVERABLE	Reading buffer index %s is invalid
4901	RECOVERABLE	The maximum index for this buffer is %d
4902	RECOVERABLE	Reading buffers must be able to contain at least one element
4903	RECOVERABLE	Reading buffer expired
4904	SERIOUS	ICX parameter count mismatch, %s (Line #%d)
4905	SERIOUS	ICX parameter invalid value, %s (Line #%d)
4906	SERIOUS	ICX invalid function id, %s (Line #%d)
5000	FATAL	Bad SMU AFPGA Image size
5001	FATAL	SMU is unresponsive
5002	FATAL	Communication Timeout with DFPGA
5003	SERIOUS	Saved calibration constants corrupted
5004	SERIOUS	Operation conflicts with CALA sense mode
5005	SERIOUS	Value too big for range
5006	SERIOUS	Value too small for range
5007	SERIOUS	Operation would exceed safe operating area of the instrument
5008	SERIOUS	Operation not permitted while output is on
5009	SERIOUS	Unknown sourcing function
5010	SERIOUS	No such SMU function
5011	SERIOUS	Operation not permitted while cal is locked
5012	SERIOUS	Cal data not saved - save or restore before lock
5013	SERIOUS	Cannot save cal data - unlock before save
5014	SERIOUS	Cannot restore cal data - unlock before restore
5015	SERIOUS	Save to cal set disallowed
5016	SERIOUS	Cannot change cal date - unlock before operation
5017	SERIOUS	Cannot change cal constants - unlock before operation
5018	SERIOUS	Cal version inconsistency
5019	SERIOUS	Cannot unlock - invalid password
5020	SERIOUS	Error saving cal data to non-volatile storage
5021	SERIOUS	Cannot restore default calset. Using previous calset

表 B-2 （続き）
エラー一覧

エラー番号	エラーレベル	エラーメッセージ
5022	SERIOUS	Cannot restore previous calset. Using factory calset
5023	SERIOUS	Cannot restore factory calset. Using nominal calset
5024	SERIOUS	Cannot restore nominal calset. Using firmware defaults
5025	SERIOUS	Cannot set filtercount > 1 when measure.count > 1
5026	SERIOUS	Cannot set measure.count > 1 when filtercount > 1
5027	SERIOUS	Unlock cal data with factory password
5028	SERIOUS	Cannot perform requested operation while source autorange is enabled
5029	SERIOUS	Cannot save without changing cal date and cal due values
5030	SERIOUS	Not implemented yet
5031	RECOVERABLE	Cannot modify built-in reading buffers
5032	RECOVERABLE	Cannot change this setting unless buffer is cleared
5033	RECOVERABLE	Reading buffer not found within device
5034	RECOVERABLE	No readings exist within buffer
5035	RECOVERABLE	Table not found within buffer
5036	RECOVERABLE	Attribute not found
5037	RECOVERABLE	Cannot delete NVRAM Reading Buffer
5038	RECOVERABLE	Index exceeds maximum reading
5039	RECOVERABLE	Measure count exceeds buffer capacity
5040	RECOVERABLE	Cannot use same reading buffer for multiple overlapped measurements
5041	SERIOUS	Output Enable not asserted
5042	SERIOUS	Invalid while overlapped measure
5043	SERIOUS	Cannot perform requested operation while voltage measure autorange is enabled
5044	SERIOUS	Cannot perform requested operation while current measure autorange is enabled
5045	SERIOUS	Cannot perform requested operation while filter is enabled
5046	SERIOUS	SMU too hot
5047	RECOVERABLE	Minimum timestamp resolution is 1µs.

C
共通コマンド

付録 C のトピックス

はじめに , C-2

共通コマンド , C-2
コマンド要約 , C-2
対応するスクリプトコマンド , C-3
コマンドリファレンス , C-4

はじめに

この付録では共通コマンドと、それに対応するスクリプトコマンドについて解説します。

共通コマンド

コマンド要約

260x 型ソースメータがサポートする共通コマンドを表 C-1 に示します。この表ではコマンドを大文字で表記していますが、共通コマンドは大文字と小文字の区別はされませんので大文字と小文字どちらを使ってもかまいません。これらのコマンドは IEEE- 488.2 規格が制定しているコマンドと基本的に同じものですが、260x 型は必ずしも厳密にこの規格に準拠していない個所がありますので注意してください。

表 C-1
共通コマンド

ニモニック	名称	コマンド要約
*CLS	Clear status	すべてのイベントレジスタとエラーキューをクリアします。[1]
*ESE <mask>	Event enable command	標準イベントイネーブルレジスタをプログラムします。[1]
*ESE?	Event enable query	標準イベントイネーブルレジスタを読み出します。[1]
*ESR?	Event status register query	標準イベントイネーブルレジスタを読み出し / クリアします。[1]
*IDN?	Identification query	製造者名、モデル型式、シリアル番号、およびファームウェアのバージョン番号を返します。
*OPC	Operation complete command	保留中のすべてのコマンドが実行されるのを待ち、標準イベントレジスタの操作完了ビットをセットします。
*OPC?	Operation complete query	選択したデバイスの保留中であった操作が完了するのを待ち、ASCII 文字 “1” を出力キューに書き込みます。
*RST	Reset command	ソースメータをデフォルト設定状態に戻します。
*SRE <mask>	Service request enable command	サービス要求イネーブルレジスタをプログラムします。[1]
*SRE?	Service request enable query	サービス要求イネーブルレジスタを読み出します。[1]
*STB?	Status byte query	ステータスバイトレジスタの内容を読み出します。[1]
*TRG	Trigger command	ソースメータへリモートトリガを送ります。
*TST?	Self-test query	0 を返します。
*WAI	Wait-to-continue command	それ以前に実行開始したすべてのコマンドの完了を待ちます。

1. ステータスコマンドについては付録 D の説明をご覧ください。

対応するスクリプトコマンド

表 C-1 に記載の共通コマンドに対応するスクリプトコマンドをまとめて表 C-2 に示します。スクリプトコマンドについて詳しくはセクション 12 をご覧ください。

表 C-2
スクリプトコマンドとの対応

共通コマンド	対応するスクリプトコマンド
*CLS	status.reset()
*ESE?	print(tostring(status.standard.enable))
*ESE <mask>	status.standard.enable = <mask>
*ESR?	print(tostring(status.standard.event))
*IDN?	print([[Keithley Instruments Inc., Model]]..localnode.model.. [[,]]..localnode.serialno..[[,]]..localnode.revision)
*OPC?	waitcomplete() print([[1]])
*OPC	opc()
*RST	reset()
*SRE?	print(tostring(status.request_enable))
*SRE <mask>	status.request_enable = <mask>
*STB?	print(tostring(status.condition))
*TRG	N/A
*TST?	print([[0]])
*WAI	waitcomplete()

コマンドリファレンス

ステータスモデルに関連するコマンドを除くすべての共通コマンドの詳細を以下に説明します。ステータスコマンドの使用法については付録 D をご覧ください。

*IDN? — identification query　　ID 情報を読み取ります

識別用の文字列には製造メーカー名、モデル型式、シリアル番号、ファームウェアの改訂レベルが含まれており、次に示す形式で送信されます。

Keithley Instruments Inc., Model nnnn, xxxxxxx, yyyyy

ここに、 nnnn はモデル番号（Model 2601 または Model 2602）、
xxxxxxx はシリアル番号、
yyyyy はファームウェアの改訂レベルを表します。

*OPC — operation complete　　OPC ビットをセットします
*OPC? — operation complete query　出力キューに “1” を書き込みます

*OPC が送信されると、その時点でオーバーラップしているコマンドがあればその完了を待って、標準イベントレジスタ（付録 D 参照）の OPC ビットがセットされます。出力キューにも ASCII “1” が書き込まれ、オーバーラップしているコマンドが終了したときに、クエリ *OPC? によって読み出されます。

*RST — reset　　ソースメータをデフォルトセットアップの状態に戻します

*RST コマンドが送られると、ソースメータはデフォルト条件（表 1-3、p.1-24）に復帰します。

*TRG — trigger　　ソースメータにリモートトリガを送ります

*TRG コマンドを使用してソースメータへ GPIB/RS-232 トリガを送ります。グループがトリガを実行（GET）するのと同じ効果を持ちます。

*TST? — self-test query　　0 を返します

このコマンドは常に 0 を出力キューに書き込みます。このコマンドは互換性維持のために共通コマンドに含められていますが、260x 型は実際には自己テストを実行しません。

*WAI — wait-to-continue コマンドの実行完了を待ちます

デバイスコマンドに 2 つのタイプが存在します。

- 順次実行コマンド — 次のコマンド実行前に操作を完了することが許されたコマンド。
- オーバーラップ コマンド — それ以前に実行を開始したオーバーラップコマンドによるデバイス操作がまだ未完了で、次に続くコマンドの実行が許されます。

それ以前に起動されたすべてのオーバーラップ コマンドによる装置操作の完了を待ってから次のコマンドを起動したい場合に *WAI コマンドを使用します。したがって、順次実行コマンドは *WAI を必要としません。

D
ステータスモデル

付録 D のトピックス

概要 , D-2
- ステータスバイトと SRQ, D-2
- ステータスレジスタセット , D-2
- キュー , D-2
- ステータス関数のまとめ , D-8

レジスタとキューのクリア , D-9

レジスタのプログラミングと読み出し , D-10
- イネーブルレジスタと遷移レジスタのプログラミング , D-10
- レジスタの読み出し , D-11

ステータスバイトとサービス要求（SRQ）, D-11
- ステータスバイトレジスタ , D-11
- サービス要求イネーブルレジスタ , D-13
- シリアルポーリングと SRQ, D-14
- SPE、SPD (シリアルポーリング), D-14
- ステータスバイトとサービス要求コマンド , D-14
- イネーブルレジスタと遷移レジスタ , D-15
- ノードと SRQ イネーブルレジスタのコントロール , D-15

ステータスレジスタセット , D-17
- システムサマリイベントレジスタ（System Summary Event Register）, D-17
- 標準イベントレジスタ（Standard Event Register）, D-19
- 動作イベントレジスタ（Operation Event Register）, D-21
- クエスチョナブルイベントレジスタ（Questionable Event Register）, D-24
- 測定イベントレジスタ（Measurement Event Register）, D-26
- レジスタのプログラミング例 , D-29

キュー , D-29
- 出力キュー , D-29
- エラーキュー , D-29

概要

オペレータが装置に発生しているイベントをモニタして操作できるようにするため、ソースメータは一連のステータスレジスタとキューを備えています。ステータスの構成を図 D-1 から 図 D-5 までに示します。この中でステータスモデルの中心的な役割を果たすのがステータスバイトレジスタ（Status Byte Register）です。このレジスタをユーザのテストプログラムで読み出すことによって、サービス要求（SRQ）発生の有無とその原因を知ることができます。

ステータスバイトと SRQ

ステータスバイトレジスタは 5 個のステータスレジスタセットと 2 個のキューの内容をまとめたサマリビットを受け取ります。このレジスタセットとキューには装置の様々なイベントをモニタできる情報が含まれています。イネーブルされているイベントが発生すると、それによってステータスバイトレジスタのサマリビットがセットされます。ステータスバイトのサマリビットがセットされ、それに対応するイネーブルビットをセット（ユーザがプログラムします）すると、RQS/MSS ビットがセットされて SRQ の発生を示すとともに、GPIB の SRQ ラインがアサートされます。

ステータスレジスタセット

標準的なステータスレジスタセットは条件レジスタとイベントレジスタ、およびイベントイネーブルレジスタによって構成されます。（多くはこの他に負極性と正極性遷移を表すレジスタを備えています。）条件レジスタは装置の現在の動作条件を示すために絶えず内容が更新されている読み出し専用レジスタです。

あるイベントが起こると、該当するイベントレジスタのビットが 1 にセットされ、このビットはレジスタがリセットされるまで値が 1 にラッチされています。あるイベントレジスタビットがセットされ、それに対応するイネーブルビットがセット（ユーザによるプログラム）されると、そのレジスタの出力（サマリ）が 1 にセットされます。その結果として低レベルレジスタの別なビットがセットされて、最終的にステータスバイトレジスタのサマリビットがセットされます。

キュー

ソースメータは出力キュー（Output Queue）とエラーキュー（Error Queue）を使用します。応答メッセージ、たとえば要求されている読み取り値は出力キューに書き込まれます。各種のプログラミングエラーやステータスメッセージは発生する都度エラーキューに書き込まれてゆきます。キュー内にデータが存在している間、ステータスバイトレジスタの該当サマリビットがセットされています（エラーキューの EAV、出力キューの MAV）。

図 D-1
ステータスの概要

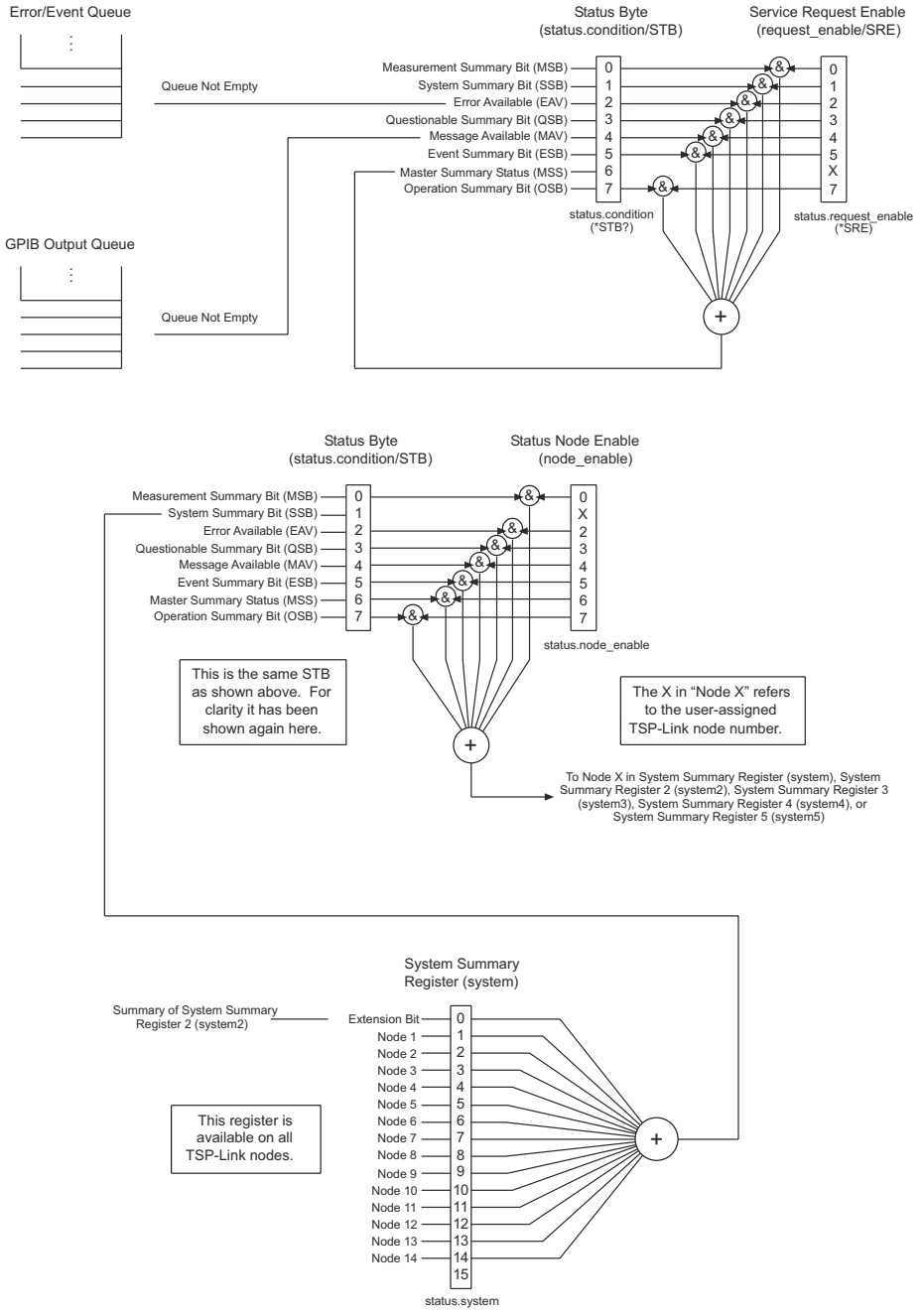


図 D-2
ステータスモデル（システムサマリと標準イベントレジスタ）

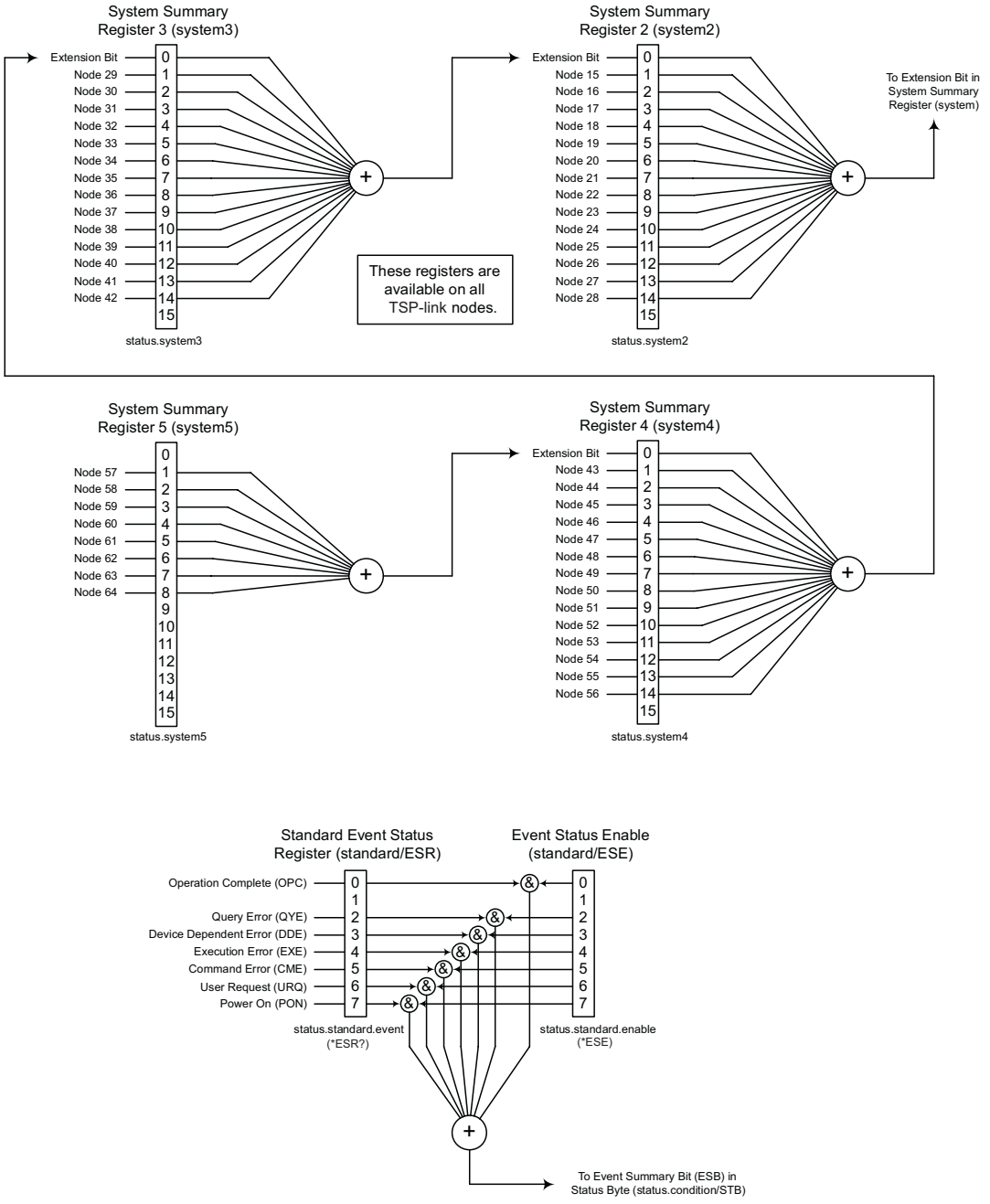


図 D-3
ステータスモデル（動作イベントレジスタ）

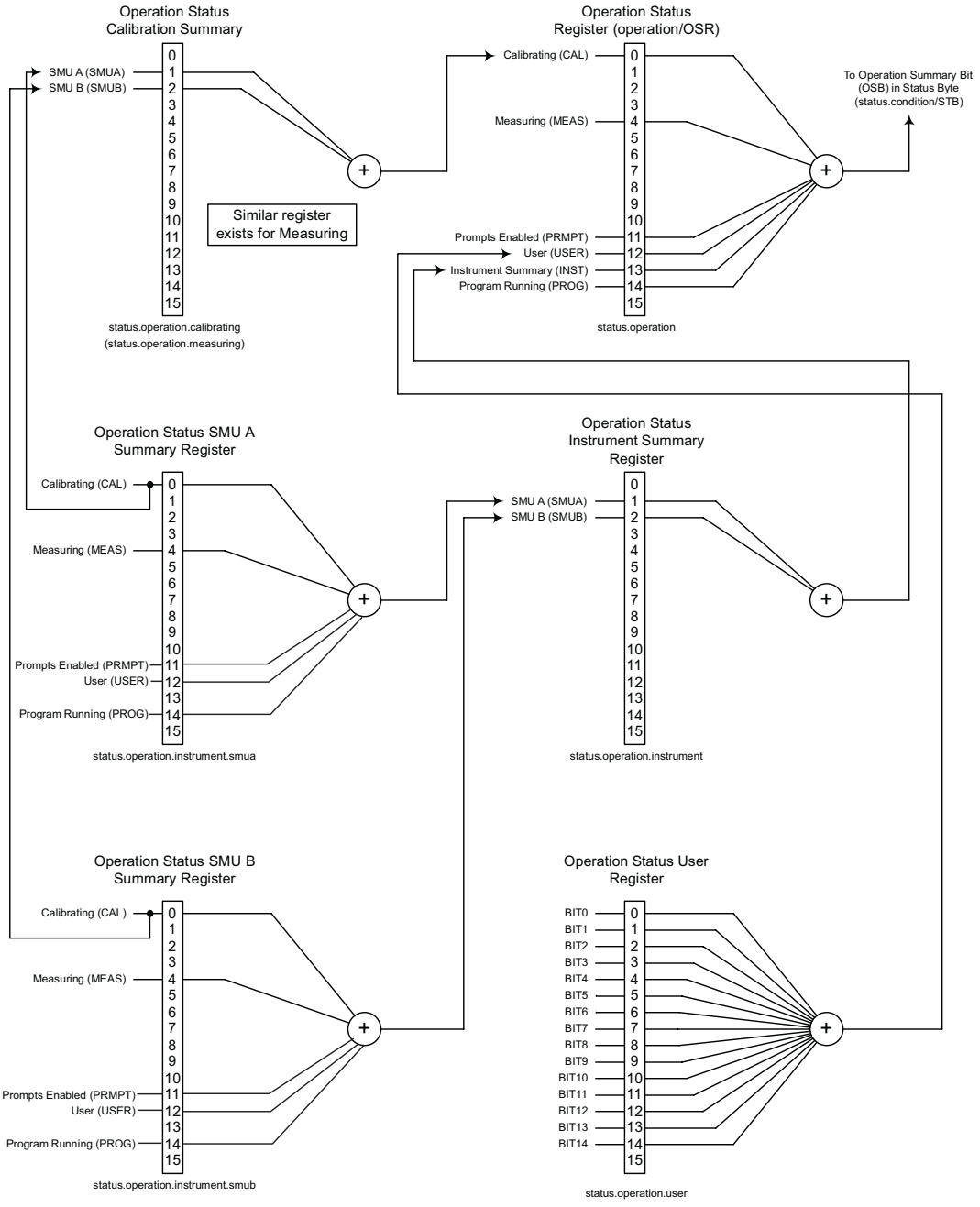


図 D-4
ステータスモデル（クエスチョナブルイベントレジスタ）

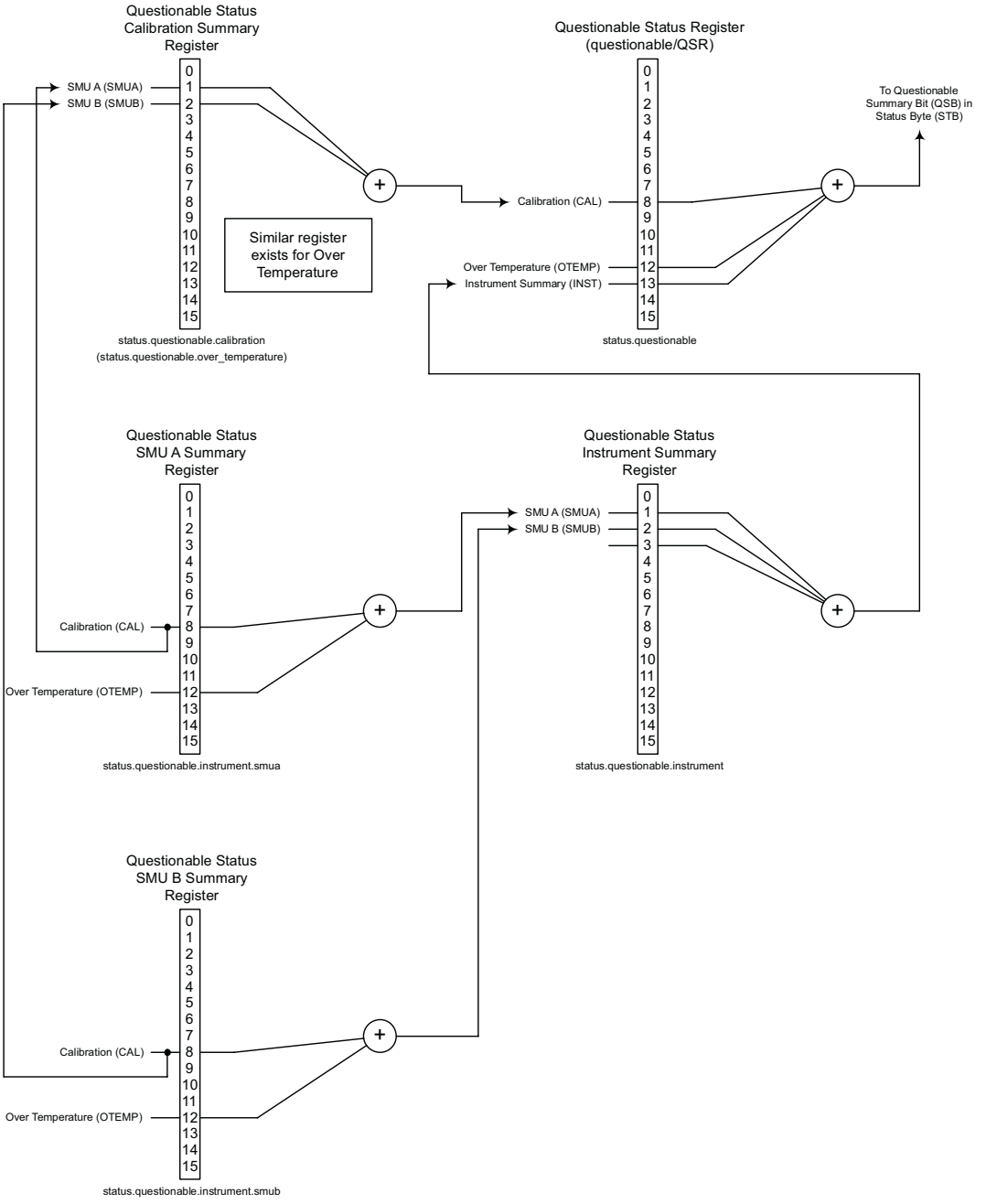
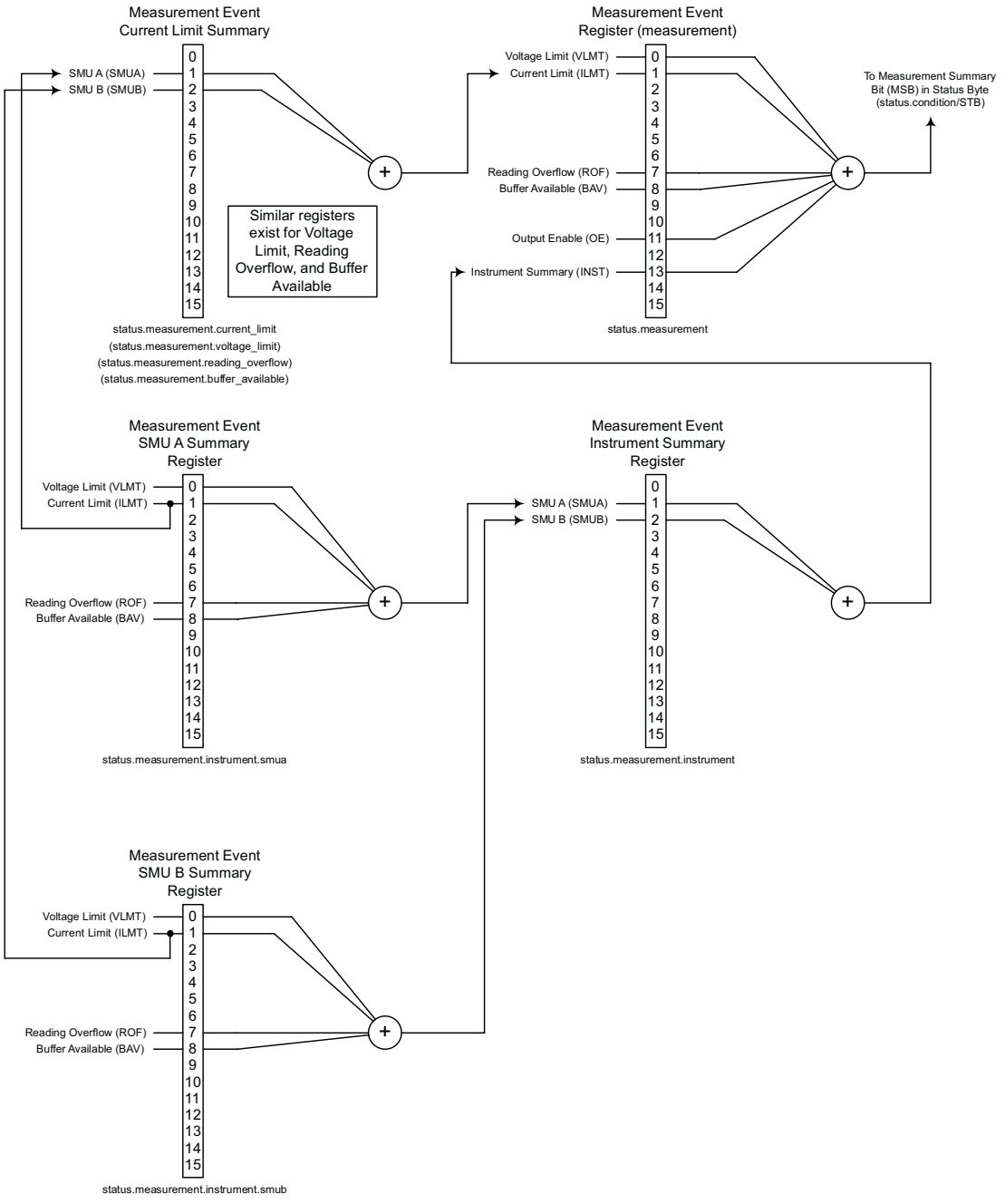


図 D-5
ステータスモデル（測定イベントレジスタ）



ステータス関数のまとめ

以下の関数群を使用して各種レジスタのコントロールと読み出しを行います（表 D-1）。共通ステータスコマンド一覧が付録 C にまとめられています。また、各種レジスタセットに関連するコマンド一覧についてはこのセクションの後半をご覧ください。

表 D-1
ステータス関数とレジスタ

タイプ	関数 [1]	参照項目
システムサマリ	status.reset status.node_enable status.request_enable status.node_event status.request_event status.condition	Page D-15
測定イベント	status.measurement.* status.measurement.instrument.smuX.* status.measurement.instrument.* status.measurement.voltage_limit.* status.measurement.current_limit.* status.measurement.buffer_available.* status.measurement.reading_overflow.*	Page D-26
動作イベント	status.operation.* status.operation.instrument.smuX.* status.operation.instrument.* status.operation.calibrating.* status.operation.measuring.* status.operation.user.*	Page D-21
クエスチョナブルイベント	status.questionable.* status.questionable.instrument.smuX.* status.questionable.instrument.* status.questionable.calibration.* status.questionable.over_temperature.*	Page D-24
標準イベント	status.standard.enable	Page D-19

1. * = '.ntr'、'.ptr'、および '.enable'；smuX = smua または smub.

表 D-1 (続き)
ステータス関数とレジスタ

タイプ	関数[1]	参照項目
システムイベント	status.system.enable status.system2.enable status.system3.enable status.system4.enable status.system5.enable	Page D-17

レジスタとキューのクリア

ソースメータへ電源を入れると、各種レジストのステータスを表す要素は次のようにセットされます。

- status.condition レジスタの PON ビットがセットされます
- インターロックや過温度ビットなどが目的に応じて適切にセットされます
- イネーブルレジスタはすべて 0 にセットされます
- NTR レジスタはすべて 0 にセットされます
- 使用されている PTR レジスタはすべて 1 にセットされます
- 2 個のキューが空になります

ステータスレジスタとエラーキューをリセットするコマンド群を表 D-2 に示します。これらのコマンドに加えて、プログラム可能なレジスタであれば個々のコマンドに引数 0 を与えてプログラムすることによりリセットが可能です。

表 D-2
レジスタのリセット、キューのクリアに使用するコマンド

コマンド	説明
レジスタをリセットします。	
*CLS または status.reset()	ステータスレジスタのビットを 0 へリセットします。
エラーキューをクリアします。	
errorqueue.clear()	エラーキューから全てのメッセージを消去します。

レジスタのプログラミングと読み出し

イネーブルレジスタと遷移レジスタのプログラミング

ユーザがプログラムできるレジスタはイネーブルレジスタと遷移レジスタだけです。ステータス構成に含まれる他のすべてのレジスタは読み取り専用です。これ以降、まずイネーブルレジスタのプログラミングに使用する各種コマンドに与えるパラメータ値を確定させる方法について説明し、実際のコマンドについては付録 C、および表 D-1（p.D-8）でまとめて解説します。

イベントイネーブルまたは遷移レジスタをプログラミングするコマンドを送信するときは、該当するレジスタの各ビットを希望する状態（0 または 1）に指定するパラメータ値を与える必要があります。レジスタのビット位置（図 D-6）は 2 進数でのパラメータ値とそれに対応する 10 進数値を示します。これらのレジスタのいずれかをプログラムするには、セットしたいビット（複数可）を 10 進数で表現した値を送ります。レジスタについては「イネーブルレジスタと遷移レジスタ」（p.15）で更に詳しく記されています。また、これらのレジスタのプログラムに使用する特定のコマンドについては、このセクションの後半に簡単な説明があります。

図 D-6
16-bit ステータスレジスタ

Bit Position	B7	B6	B5	B4	B3	B2	B1	B0
Binary Value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal Weights	128 (2^7)	64 (2^6)	32 (2^5)	16 (2^4)	8 (2^3)	4 (2^2)	2 (2^1)	1 (2^0)

A. Bits 0 through 7

Bit Position	B15	B14	B13	B12	B11	B10	B9	B8
Binary Value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal Weights	32768 (2^{15})	16384 (2^{14})	8192 (2^{13})	4096 (2^{12})	2048 (2^{11})	1024 (2^{10})	512 (2^9)	256 (2^8)

B. Bits 8 through 15

数値パラメータを使用するときは、たとえば下の例のような適当なマスク（<mask>）を使用してレジスタをプログラムすることができます。

```
*ese <mask>
status.standard.enable = <mask>
```

10 進数を 2 進数へ変換するときは図 D-6 に示す情報を利用するのが便利です。たとえば、B0、B4、B7 および B10 ビットをセットしたいとすれば、これに対応する 10 進数 1169 をマスクパラメータ（1169 = 1 + 16 + 128 + 1024）として使用できます。

レジスタの読み出し

ステータス構成に含まれるいずれのレジスタも、共通コマンドによる問合せ（該当する場合のみ）を使用するか、または print() / print(tostring()) コマンドに目的とするレジスタ用のスクリプトコマンドを組み合わせることで読み出しが可能です。print() コマンドは数値を返し、print(tostring()) コマンドはそれに対応する文字列を返します。たとえば、以下に示すコマンドはいずれもサービス要求イネーブルレジスタの値を要求します。

```
*SRE?
print(tostring(status.request_enable))
print(status.request_enable)
```

応答メッセージはレジスタのどのビットがセットされているかを示す 10 進数を返します。たとえば、返り値が 10 進数 37 であったとすれば、図 D-6 を参照してこの値を 2 進数へ変換すると 100101 が得られますから、B5、B2、および B0 がセットされていることが分ります。

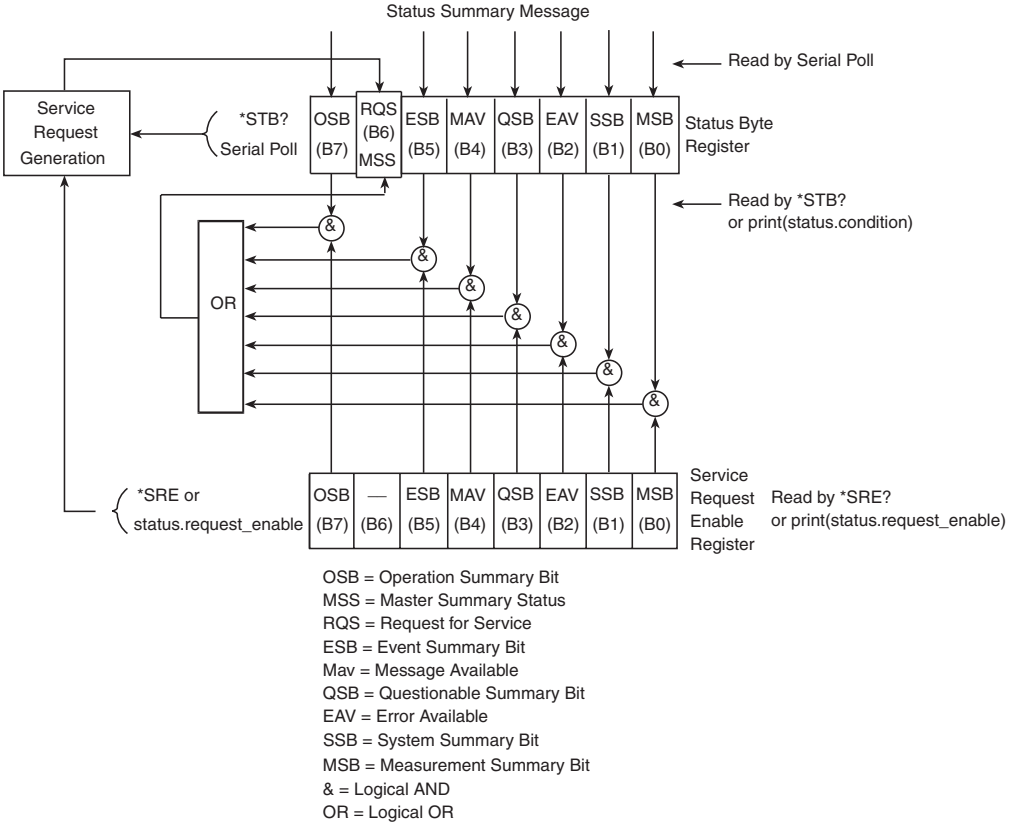
ステータスバイトとサービス要求（SRQ）

サービス要求のコントロールにはステータスバイトレジスタ（Status Byte Register）とサービスリクエストイネーブルレジスタ（Service Request Enable Register）という 2 つの 8 ビットレジスタを使用します。これらのレジスタの構成を図 D-7 に示します。

ステータスバイトレジスタ

ステータスレジスタとキューから取得したサマリメッセージを使用して、ステータスバイトレジスタ（Status Byte Register）の該当するビット（B0、B1、B2、B3、B4、B5、および B7）をセット / クリアします。これらのサマリビットはラッチされませんから、その状態（0/1）はサマリメッセージ（0/1）の内容だけによって決定されます。たとえば、標準イベントレジスタ（Standard Event Register）が読み込まれると、そのレジスタはクリアされます。この結果、サマリメッセージは 0 にリセットされ、それによってステータスバイトレジスタの ESB ビットがリセットされます。

図 D-7
ステータスバイトとサービス要求（SRQ）



ステータスバイトレジスタ（Status Byte Register）の各ビットの意味は次の通りです。

- **ビット B0、測定サマリビット**（MSB）— イネーブルされた測定イベントが発生すると、このサマリビットがセットされます。
- **ビット B1、システムサマリビット**（SSB）— イネーブルされたシステムイベントが発生すると、このサマリビットがセットされます。
- **ビット B2、エラーアベイラブル**（EAV）— エラーキューにエラー / ステータスメッセージが書き込まれると、このビットがセットされます。
- **ビット B3、クエスチョナブルサマリビット**（QSB）— イネーブルされたクエスチョナブルイベントが発生すると、このサマリビットがセットされます。
- **ビット B4、メッセージアベイラブル**（MAV）— 出力キューに応答メッセージが存在すると、このビットがセットされます。
- **ビット B5、イベントサマリビット**（ESB）— イネーブルされた標準イベントが発生すると、このサマリビットがセットされます。
- **ビット B6、サービス要求**（RQS）**/ マスタサマリステータス**（MSS）— ステータスバイトレジスタのイネーブルされたサマリビットがセットされると、このビットがセットされます。ステータスバイトレジスタのビット B6 は使い方によってサービス要求（RQS）ビット、またはマスタサマリステータス（MSS）ビットのいずれかの意味を持ちます。
  - ソースメータの GPIB シリアルポールシーケンスを使用してステータスバイト（別名：シリアルポールバイト）を取得しようとするとき、B6 は RQS ビットとして機能します。シリアルポールシーケンスの使用法について詳しくは「シリアルポーリングと SRQ」をご覧ください。
  - 共通コマンド *STB? または status.condition（表 D-3、p.D-15）を使用してステータスバイトを読み取ろうとするとき、B6 は MSS ビットとして機能します。
- **ビット B7、動作サマリ**（OSB）— イネーブルされた動作イベントが発生すると、このサマリビットがセットされます。

サービス要求イネーブルレジスタ

サービス要求の発生はサービス要求イネーブルレジスタ（Service Request Enable Register）によってコントロールされます。このレジスタはユーザによってプログラムされ、ステータスバイトレジスタのステータスサマリメッセージビット（B0、B2、B3、B4、B5、および B7）によってビット B6（RQS/MSS）が設定されるのを許すか否かを指定します。図 D-7 に示すように、サマリビット（複数）とサービス要求イネーブルレジスタの対応するイネーブルビットとの間の論理積（AND、&）が評価されます。セット（1）されたサマリビットとイネーブルレジスタのイネーブルされたビット（1）の論理積がとられたとすれば、両者の論理積（1）が OR ゲートに入力され、したがってステータスバイトレジスタの MSS/RQS ビットがセットされます。

サービス要求イネーブルレジスタの個々のビットは、共通コマンド *SRE またはこれに相当するスクリプトを使用してセット / クリアすることができます。サービス要求イネーブルレジスタの読み出しにはクエリコマンド *SRE?、またはこれに相当するスクリプトを使用します。サービス要求イネーブルレジスタは電源を一度切ってから再投入するか、またはパラメータ 0 を与えて *SRE コマンドを実行（つまり *SRE 0）するとクリアされます。SRQ イネーブルレジスタのプログラミングと読み出しに使用するコマンドをまとめて表 D-3（p.D-15）に示します。

シリアルポーリングと SRQ

イネーブルされているイベントサマリビットのいずれかが 0 から 1 へ変化すると、B6 がセットされて SRQ（サービス要求）が発生します。ユーザはプログラムの中で定期的にステータスバイトを読むことによって、SRQ 発生の有無とその発生源をチェックすることができます。SRQ の発生を検出したときに適当なサブルーチンに分岐するようにプログラムしておけば、その要求を適正に処理することができます。

ソースメータ側のシリアルポーリングシーケンスを使用して SRQ を管理することも可能です。SRQ が発生しなければステータスバイトレジスタの B6（RQS）はクリアされたままですから、プログラムはシリアルポール実行後も単純に通常の処理を継続します。SRQ が発生するとステータスバイトレジスタのビット B6 がセットされますから、シリアルポーリングが SRQ を検出したときにサブルーチンへ分岐してサービスを行うようにプログラムしておきます。

シリアルポールを実行するとステータスバイトレジスタの RQS は自動的にリセットされます。したがって、それ以後実行するシリアルポールで B6 を常にモニタしていれば、他のイベントタイプが発生させた SRQ を検出することができます。

共通コマンドとスクリプトコマンドでは、B6 が MSS（Message Summary Status）ビットとなります。シリアルポールは MSS をクリアしません。したがって、MSS ビットはすべてのステータスバイトサマリビットがリセットされるまでセットされた状態を保ちます。

SPE、SPD (シリアルポーリング)

GPIB インターフェイスに限定すると、ソースメータ側でのシリアルポーリング実行には SPE、SPD 一般バスコマンド（General Bus Command）が使用されます（「一般バスコマンド」、p.11-7、参照）。このシリアルポーリングによりシリアルポールバイト（ステータスバイト）が得られます。標準的にはシリアルポーリングを実行するのはコントローラであり、その目的は複数の装置のどれが SRQ ラインにサービス要求を出しているのかを決定することです。

ステータスバイトとサービス要求コマンド

ステータスバイトレジスタ（Status Byte Register）とサービス要求イネーブルレジスタ（Service Request Enable Register）のプログラムと読み出しに使用するコマンドを表 D-3 に示します。この表には共通コマンドとそれに対応するスクリプトコマンドの両方が記載されています。レジスタのプログラミングと読み出しに関する更に詳しい説明については「レジスタのプログラミングと読み出し」（p.D-10）および「レジスタの読み出し」（p.D-11）をご覧ください。

サービス要求イネーブルレジスタのビットを 0 にリセットするには、パラメータとして 0 を与えてコマンドを実行します（たとえば、*SRE 0）。

表 D-3
ステータスバイトとサービス要求イネーブルレジスタコマンド

コマンド	説明
*STB? または print(status.condition)	ステータスバイトレジスタを読み出します。
*SRE <mask> または status.request_enable = <mask>	サービス要求イネーブルレジスタのプログラム： <mask> = 0 to 255
*SRE? または print(status.request_enable)	サービス要求イネーブルレジスタを読み出します。

イネーブルレジスタと遷移レジスタ

一般に、ユーザが書き込み可能なレジスタには 3 種類があり、これらのレジスタを使用してどのビットをレジスタサマリに含めるか、どのタイミングでそれが起るかを設定します。これらのレジスタはコマンド一覧表の注記で次のように区別されています。

- イネーブルレジスタ（表脚注に “enable”という注記あり）: 各種関連イベントをそのレジスタのサマリビットに含めることができます。
- ネガティブ遷移レジスタ（NTR: 表脚注に “ntr”という注記あり）: NTR のあるビットがセットされるとイベントレジスタの特定のビットがセットされ、それに対応する条件レジスタのビットが 1 から 0 へ変化します。
- ポジティブ遷移レジスタ（PTR: 表脚注に “ptr”という注記あり）: PTR のあるビットがセットされるとイベントレジスタ内の特定のビットがセットされ、それに対応する条件レジスタのビットが 0 から 1 へ変化します。

ノードと SRQ イネーブルレジスタのコントロール

システムノードと SRQ イネーブルビットをコントロールする属性値、および関連レジスタの読み取りに係わる属性値を表 D-4 にまとめて示します。たとえば、次の 2 つはいずれもシステムノードの MSB イネーブルビットをセットします。

```
status.node_enable = status.MSB
status.node_enable = 1
```

表 D-4
システムノードと SRQ イネーブルレジスタビットに関連する属性値

属性値	機能	ビット
システムノード イネーブルレジスタのビットをセットします。		
status.node_enable = status.MEASUREMENT_SUMMARY_BIT	Enable MSB.	B0
status.node_enable = status.MSB	Enable MSB.	B0
status.node_enable = status.ERROR_AVAILABLE	Enable EAV bit.	B2
status.node_enable = status.EAV	Enable EAV bit.	B2
status.node_enable = status.QUESTIONABLE_SUMMARY_BIT	Enable QSB.	B3
status.node_enable = status.QSB	Enable QSB.	B3
status.node_enable = status.MESSAGE_AVAILABLE	Enable MAV bit.	B4
status.node_enable = status.MAV	Enable MAV bit.	B4
status.node_enable = status.EVENT_SUMMARY_BIT	Enable ESB.	B5
status.node_enable = status.ESB	Enable ESB.	B5
status.node_enable = status.MASTER_SUMMARY_STATUS	Enable MSS bit.	B6
status.node_enable = status.MSS	Enable MSS bit.	B6
status.node_enable = status.OPERATION_SUMMARY_BIT	Enable OSB.	B7
status.node_enable = status.OSB	Enable OSB.	B7
サービス要求イネーブルレジスタのビットをセットします。		
status.request_enable = status.MEASUREMENT_SUMMARY_BIT	Enable MSB.	B0
status.request_enable = status.MSB	Enable MSB.	B0
status.request_enable = status.SYSTEM_SUMMARY_BIT	Enable SSB.	B1
status.request_enable = status.SSB	Enable SSB.	B1
status.request_enable = status.ERROR_AVAILABLE	Enable EAV bit.	B2
status.request_enable = status.EAV	Enable EAV bit.	B2
status.request_enable = status.QUESTIONABLE_SUMMARY_BIT	Enable QSB.	B3
status.request_enable = status.QSB	Enable QSB.	B3
status.request_enable = status.MESSAGE_AVAILABLE	Enable MAV bit.	B4
status.request_enable = status.MAV	Enable MAV bit.	B4
status.request_enable = status.EVENT_SUMMARY_BIT	Enable ESB.	B5
status.request_enable = status.ESB	Enable ESB.	B5
status.request_enable = status.OPERATION_SUMMARY_BIT	Enable OSB.	B7
status.request_enable = status.OSB	Enable OSB.	B7
レジスタを読み出します。		
print(status.node_enable)	システムイネーブルレジスタを要求	
print(status.request_enable)	SRQイネーブルレジスタを要求	
print(status.condition)	ステータスバイトレジスタを要求	

ステータスレジスタセット

図 D-1 から図 D-5 までに示すようにソースメータのステータス構造には次の 5 種類のステータスレジスタセットが含まれています。System Summary Event Status、Standard Event Status、Operation Event Status、Measurement Event Status、および Questionable Event Status。

システムサマリイベントレジスタ（System Summary Event Register）

図 D-1（p.D-3）と図 D-2（p.D-4）が示すように、システムイベントステータス（System Event Status）に関連する 5 個のレジスタセットが存在します。これらのレジスタは TSP-Link（セクション 9 参照）に接続された各種ノードのシステムステータスを要約した情報を保持します。TSP-Link 上のすべてのノードは、TSP-Link が初期化されるときにシステムサマリレジスタのコピーを共有することに注意してください。これにより、すべてのノードから他のノードのステータスモデル（SRQ を含む）へのアクセスが可能になります。

関連するコマンドを表 D-5 にまとめて示します。

たとえば、次の 2 つのコマンドはいずれも EXT イネーブルビットをセットします。

```
status.system.enable = status.system.EXT
status.system.enable = 1
```

次のコマンドはシステムイネーブルレジスタの値を要求します。

```
print(status.system.enable)
```

システムイベントレジスタで使用されるビットは次の意味を持ちます。

- **EXT** — このビットがセットされているときは、別なシステムステータスレジスタからの拡張ビットがセットされています。
- **NODEn** — TSP-Link ノード n 上のビットがセットされていることを示します（n=1-64）。

表 D-5
システムサマリイベントに関連するコマンド

コマンド[1]	ビット[1]
レジスタのビットをセットします。	
status.system.enable = status.system.EXTENSION_BIT	B0
status.system.enable = status.system.EXT	B0
status.system.enable = status.system.NODEn	Bn
レジスタを読み出します。	
print(status.system.enable)	
print(status.system.condition)	
print(status.system.event)	

1. ステータスシステム：n=1 ～ 14、ステータスシステム 2：n=15 ～ 28、ステータスシステム 3：n=29 ～ 42、ステータスシステム 4：n=43 ～ 56、ステータスシステム 5：n=57 ～ 64

表 D-5 (続き)
システムサマリイベントに関連するコマンド

コマンド[1]	ビット[1]
レジスタのビットをセットします。 status.system2.enable = status.system2.EXTENSION_BIT status.system2.enable = status.system2.EXT status.system2.enable = status.system2.NODEn レジスタを読み出します。 print(status.system2.enable) print(status.system2.condition) print(status.system2.event)	 B0 B0 Bn
レジスタのビットをセットします。 status.system3.enable = status.system3.EXTENSION_BIT status.system3.enable = status.system3.EXT status.system3.enable = status.system3.NODEn レジスタを読み出します。 print(status.system3.enable) print(status.system3.condition) print(status.system3.event)	 B0 B0 Bn
レジスタのビットをセットします。 status.system4.enable = status.system4.EXTENSION_BIT status.system4.enable = status.system4.EXT status.system4.enable = status.system4.NODEn レジスタを読み出します。 print(status.system4.enable) print(status.system4.condition) print(status.system4.event)	 B0 B0 Bn
レジスタのビットをセットします。 status.system5.enable = status.system5.EXTENSION_BIT status.system5.enable = status.system5.EXT status.system5.enable = status.system5.NODEn レジスタを読み出します。 print(status.system5.enable) print(status.system5.condition) print(status.system5.event)	 B0 B0 Bn

1. ステータスシステム：n=1 ～ 14、ステータスシステム 2：n=15 ～ 28、ステータスシステム 3：n=29 ～ 42、ステータスシステム 4：n=43 ～ 56、ステータスシステム 5：n=57 ～ 64

標準イベントレジスタ（Standard Event Register）

標準イベントレジスタ（図 D-8）の中で使用されるビットは次の意味を持ちます。

- **ビット B0、動作完了** — このビットがセットされているときは、選択したデバイスの保留 / 未完了動作がすべて完了し、ソースメータが新しいコマンドを受け付けできる状態になっていることを示します。このビットは *OPC コマンドへの応答としてセットされます。*OPC コマンドの代わりに ICL 関数 opc() を使用することも可能です。*OPC の詳細については付録 C をご覧ください。
- **ビット B1** — 未使用。
- **ビット B2、クエリエラー**（QYE）— 空の出力キューからデータを読み出そうとするとこのビットがセットされます。
- **ビット B3、デバイスに依存するエラー**（DDE）— このビットがセットされているときは、何らかの内部条件によって装置の動作が正しく実行されなかったこと示します。
- **ビット B4、実行エラー**（EXE）— ソースメータがコマンドを実行しようとして何らかのエラーを検出するとこのビットがセットされます。
- **ビット B5、コマンドエラー**（CME）— このビットはたとえば次のようなコマンドエラーの発生によってセットされます。
  - IEEE-488.2 シンタックスエラー — ソースメータが IEEE-488.2 規格に定義された構文に違反したメッセージを受信した。
  - セマンティクスエラー —ソースメータがスペルの間違いを含むコマンドを受信した、または実装されていない IEEE-488.2 オプションコマンドを受信した。
  - 装置がプログラムメッセージに埋め込まれたグループ実行トリガ（GET: Group Execute Trigger）を受信した。
- **ビット B6、ユーザ要求**（URQ）— ソースメータ前面パネルの LOCAL キーが押されるとこのビットがセットされます。
- **ビット B7、電源 ON**（PON）— このビットがセットされているときは、前回このレジスタが読み出された後でソースメータ電源の遮断と再投入が行われたことを示します。

レジスタのプログラム / 読み出しに使用するコマンドは表 D-6 にまとめられています。またビットの意味については表 D-7 をご覧ください。

図 D-8
標準イベントレジスタ

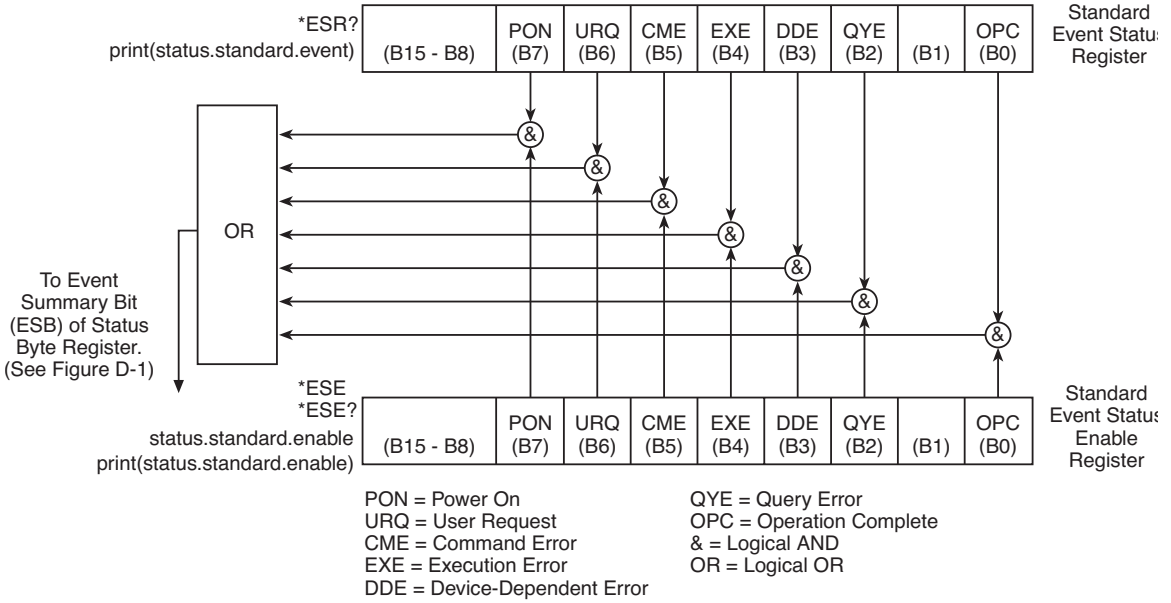


表 D-6
標準イベントコマンド

コマンド	説明
*ESR? または print(status.standard.event)	標準イベントステータスレジスタを読み出します。
*ESE <mask> または status.standard.enable = <mask>	イベントステータスイネーブルレジスタのプログラム： <mask> = 0 ～ 255、表 D-7 参照
*ESE? または print(status.standard.enable)	イベントステータスイネーブルレジスタを読み出します。

表 D-7
ステータスイベントステータスレジスタとビット

コマンド	ビット
レジスタのビットをセットします。	
status.standard.enable = status.standard.OPERATION_COMPLETE	B0
status.standard.enable = status.standard.OPC	B0
status.standard.enable = status.standard.QUERY_ERROR	B2
status.standard.enable = status.standard.QYE	B2
status.standard.enable = status.standard.DEVICE_DEPENDENT_ERROR	B3
status.standard.enable = status.standard.DDE	B3
status.standard.enable = status.standard.EXECUTION_ERROR	B4
status.standard.enable = status.standard.EXE	B4
status.standard.enable = status.standard.COMMAND_ERROR	B5
status.standard.enable = status.standard.CME	B5
status.standard.enable = status.standard.USER_REQUEST	B6
status.standard.enable = status.standard.URQ	B6
status.standard.enable = status.standard.POWER_ON	B7
status.standard.enable = status.standard.PON	B7
レジスタを読み出します。	
print(status.standard.enable)	
print(status.standard.condition)	
print(status.standard.event)	

動作イベントレジスタ（Operation Event Register）

図 D-3（p.D-5）に示すように、動作イベントステータスに関連するレジスタセットが 7 つ存在します。関連するコマンドを表 D-8 にまとめて示します。たとえば、次の 2 つのコマンドはいずれも CAL イネーブルビットをセットします。

```
status.operation.enable = status.operation.CAL
status.operation.enable = 1
```

動作イベントレジスタで使用されるビットは次の意味を持ちます。

- **CAL** — 1 つ以上のチャンネルが校正中のときにセットされます。
- **MEAS** — オーバーラップ動作による測定データ取り込み中にセットされます。通常の同期式データ取り込みではセットされません。
- **PRMPT** — コマンドプロンプトがイネーブル状態のときにセットされます。
- **USER** — 動作ステータスを表すユーザレジスタのイネーブルビットがセットされたときにセットされます。
- **INST** — 動作ステータスを表す装置サマリレジスタのイネーブルビットがセットされたときにセットされます。
- **PROG** — このビットがセットされているときはプログラムが実行中です。

表 D-8
動作イベントコマンド

コマンド[1]	ビット
レジスタのビットをセットします。	
status.operation.* = status.operation.CALIBRATING	B0
status.operation.* = status.operation.CAL	B0
status.operation.* = status.operation.MEASURING	B4
status.operation.* = status.operation.MEAS	B4
status.operation.* = status.operation.PROMPTS	B11
status.operation.* = status.operation.PRMPTS	B11
status.operation.* = status.operation.USER	B12
status.operation.* = status.operation.INSTRUMENT_SUMMARY	B13
status.operation.* = status.operation.INST	B13
status.operation.* = status.operation.PROGRAM_RUNNING	B14
status.operation.* = status.operation.PROG	B14
レジスタを読み出します。	
print(status.operation.*)	
print(status.operation.condition)	
print(status.operation.event)	
レジスタのビットをセットします。	
status.operation.instrument.smuX.* = status.operation.CALIBRATING	B0
status.operation.instrument.smuX.* = status.operation.CAL	B0
status.operation.instrument.smuX.* = status.operation.MEASURING	B4
status.operation.instrument.smuX.* = status.operation.MEAS	B4
status.operation.instrument.smuX.* = status.operation.PROMPTS	B11
status.operation.instrument.smuX.* = status.operation.PRMPTS	B11
status.operation.instrument.smuX.* = status.operation.USER	B12
status.operation.instrument.smuX.* = status.operation.PROGRAM_RUNNING	B14
status.operation.instrument.smuX.* = status.operation.PROG	B14
レジスタを読み出します。	
print(status.operation.instrument.smuX.*)	
print(status.operation.instrument.smuX.condition)	
print(status.operation.instrument.smuX.event)	
レジスタのビットをセットします。	
status.operation.instrument.* = status.operation.instrument.SMUA	B1
status.operation.instrument.* = status.operation.instrument.SMUB	B2
レジスタを読み出します。	
print(status.operation.instrument.*)	
print(status.operation.instrument.condition)	
print(status.operation.instrument.event)	

1. * = '.ntr'、'.ptr'、および '.enable'；smuX = smua または smub

表 D-8 (続き)
動作イベントコマンド

コマンド[1]	ビット
レジスタのビットをセットします。 status.operation.calibrating.* = status.operation.calibrating.SMUA status.operation.calibrating.* = status.operation.calibrating.SMUB レジスタを読み出します。 print(status.operation.calibrating.*) print(status.operation.calibrating.condition) print(status.operation.calibrating.event)	 B1 B2
レジスタのビットをセットします。 status.operation.measuring.* = status.operation.measuring.SMUA status.operation.measuring.* = status.operation.measuring.SMUB レジスタを読み出します。 print(status.operation.measuring.*) print(status.operation.measuring.condition) print(status.operation.measuring.event)	 B1 B2
レジスタのビットをセットします。 status.operation.user.* = status.operation.user.BIT0 status.operation.user.* = status.operation.user.BIT1 status.operation.user.* = status.operation.user.BIT2 status.operation.user.* = status.operation.user.BIT3 status.operation.user.* = status.operation.user.BIT4 status.operation.user.* = status.operation.user.BIT5 status.operation.user.* = status.operation.user.BIT6 status.operation.user.* = status.operation.user.BIT7 status.operation.user.* = status.operation.user.BIT8 status.operation.user.* = status.operation.user.BIT9 status.operation.user.* = status.operation.user.BIT10 status.operation.user.* = status.operation.user.BIT11 status.operation.user.* = status.operation.user.BIT12 status.operation.user.* = status.operation.user.BIT13 status.operation.user.* = status.operation.user.BIT14 レジスタを読み出します。 print(status.operation.user.*) print(status.operation.user.condition) print(status.operation.user.enable)	 B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 B10 B11 B12 B13 B14

1. * = '.ntr'、'.ptr'、および '.enable' ; smuX = smua または smub

クエスチョナブルイベントレジスタ（Questionable Event Register）

図D-4（p.D-6）に示すように、クエスチョナブルイベントステータスに関連するレジスタセットは6種類です。関連するコマンドを表D-9にまとめて示します。

たとえば、次の2つのコマンドはいずれもCALイネーブルビットをセットします。

```
status.questionable.enable = status.questionable.CAL
status.questionable.enable = 256
```

次のコマンドはクエスチョナブルイネーブルレジスタの値を要求します。

```
print(status.questionable.enable)
```

クエスチョナブルイベントレジスタで使用されるビットは次の意味を持ちます。

- **CAL** — 校正に不確かな点があるとセットされます。
- **OTEMP** — 過温度条件が検出されるとこのビットがセットされます。
- **INST** — 問題を起こしている可能性のある装置のサマリレジスタのビットがセットされると、このビットがセットされます。

表 D-9
クエスチョナブルイベントコマンド

コマンド[1]	ビット
レジスタのビットをセットします。	
status.questionable.* = status.questionable.CALIBRATION	B8
status.questionable.* = status.questionable.CAL	B8
status.questionable.* = status.questionable.OVER_TEMPERATURE	B12
status.questionable.* = status.questionable.OTEMP	B12
status.questionable.* = status.questionable.INSTRUMENT_SUMMARY	B13
status.questionable.* = status.questionable.INST	B13
レジスタを読み出します。	
print(status.questionable.*)	
print(status.questionable.condition)	
print(status.questionable.event)	
レジスタのビットをセットします。	
status.questionable.instrument.smuX.* = status.questionable.CALIBRATION	B8
status.questionable.instrument.smuX.* = status.questionable.CAL	B8
status.questionable.instrument.smuX.* = status.questionable.OVER_TEMPERATURE	B12
status.questionable.instrument.smuX.* = status.questionable.OTEMP	B12
レジスタを読み出します。	
print(status.questionable.instrument.smuX.*)	
print(status.questionable.instrument.smuX.condition)	
print(status.questionable.instrument.smuX.event)	

1. * = '.ntr'、'.ptr'、および '.enable'；smuX = smua または smub

表 D-9 (続き)
クエスチョナブルイベントコマンド

コマンド[1]	ビット
レジスタのビットをセットします。 status.questionable.instrument.* = status.questionable.instrument.SMUA status.questionable.instrument.* = status.questionable.instrument.SMUB レジスタを読み出します。 print(status.questionable.instrument.*) print(status.questionable.instrument.condition) print(status.questionable.instrument.event)	 B1 B2
レジスタのビットをセットします。 status.questionable.calibration.* = status.questionable.calibration.SMUA status.questionable.calibration.* = status.questionable.calibration.SMUB レジスタを読み出します。 print(status.questionable.calibration.*) print(status.questionable.calibration.condition) print(status.questionable.calibration.event)	 B1 B2
レジスタのビットをセットします。 status.questionable.over_temperature.* = status.questionable.over_temperature.SMUA status.questionable.over_temperature.* = status.questionable.over_temperature.SMUB レジスタを読み出します。 print(status.questionable.over_temperature.*) print(status.questionable.over_temperature.condition) print(status.questionable.over_temperature.event)	 B1 B2

1. * = '.ntr'、'.ptr'、および '.enable' ; smuX = smua または smub

測定イベントレジスタ（Measurement Event Register）

図 D-5（p.D-7）に示すように、測定イベントステータスに関連するレジスタセットは 7 種類です。関連するコマンドを表 D-10 にまとめて示します。

たとえば、次の 2 つのコマンドはいずれも VOLTAGE_LIMIT イネーブルビットをセットします。

```
status.measurement.enable = status.measurement.VOLTAGE_LIMIT
status.measurement.enable = 1
```

測定イベントレジスタで使用されるビットは次の意味を持ちます。

- **VLMT** — 電圧リミットを超過するとこのビットがセットされます。このビットの設定が更新されるのは次の 2 つのタイミングだけです。(1) 測定が行われたとき、(2) smuX.source.compliance コマンドが呼び出されたとき。
- **ILMT** — 電流リミットを超過するとこのビットがセットされます。このビットの設定が更新されるのは次の 2 つのタイミングだけです。(1) 測定が行われたとき、(2) smuX.source.compliance コマンドが呼び出されたとき。
- **ROF** — 読み込み時にオーバーフローが検出されるとセットされます。
- **BAV** — 不揮発性読み込みバッファの片方もしくは両方に少なくとも 1 個の読み取り値が存在がするとセットされます。
- **OE** — このビットがセットされているときは出力イネーブルがアサートされています。
- **INST** — 測定装置のサマリレジスタの 1 つのビットがセットされると、このビットもセットされます。

表 D-10
測定イベントコマンド

コマンド[1]	ビット
レジスタのビットをセットします。	
status.measurement.* = status.measurement.VOLTAGE_LIMIT	B0
status.measurement.* = status.measurement.VLMT	B0
status.measurement.* = status.measurement.CURRENT_LIMIT	B1
status.measurement.* = status.measurement.ILMT	B1
status.measurement.* = status.measurement.READING_OVERFLOW	B7
status.measurement.* = status.measurement.ROF	B7
status.measurement.* = status.measurement.BUFFER_AVAILABLE	B8
status.measurement.* = status.measurement.BAV	B8
status.measurement.* = status.measurement.OUTPUT_ENABLE	B11
status.measurement.* = status.measurement.OE	B11
status.measurement.* = status.measurement.INSTRUMENT_SUMMARY	B13
status.measurement.* = status.measurement.INST	B13
レジスタを読み出します。	
print(status.measurement.*)	
print(status.measurement.condition)	
print(status.measurement.event)	
レジスタのビットをセットします。	
status.measurement.instrument.smuX.* = status.measurement.VOLTAGE_LIMIT	B0
status.measurement.instrument.smuX.* = status.measurement.VLMT	B0
status.measurement.instrument.smuX.* = status.measurement.CURRENT_LIMIT	B1
status.measurement.instrument.smuX.* = status.measurement.ILMT	B1
status.measurement.instrument.smuX.* = status.measurement.READING_OVERFLOW	B7
status.measurement.instrument.smuX.* = status.measurement.ROF	B7
status.measurement.instrument.smuX.* = status.measurement.BUFFER_AVAILABLE	B8
status.measurement.instrument.smuX.* = status.measurement.BAV	B8
レジスタを読み出します。	
print(status.measurement.instrument.smuX.*)	
print(status.measurement.instrument.smuX.condition)	
print(status.measurement.instrument.smuX.event)	
レジスタのビットをセットします。	
status.measurement.instrument.* = status.measurement.instrument.SMUA	B1
status.measurement.instrument.* = status.measurement.instrument.SMUB	B2
レジスタを読み出します。	
print(status.measurement.instrument.*)	
print(status.measurement.instrument.condition)	
print(status.measurement.instrument.event)	

1. * = '.ntr'、'.ptr'、および '.enable'；smuX = smua または smub

表 D-10 (続き)
測定イベントコマンド

コマンド[1]	ビット
レジスタのビットをセットします。 status.measurement.voltage_limit.* = status.measurement.voltage_limit.SMUA status.measurement.voltage_limit.* = status.measurement.voltage_limit.SMUB レジスタを読み出します。 print(status.measurement.voltage_limit.*) print(status.measurement.voltage_limit.condition) print(status.measurement.voltage_limit.event)	 B1 B2
レジスタのビットをセットします。 status.measurement.current_limit.* = status.measurement.current_limit.SMUA status.measurement.current_limit.* = status.measurement.current_limit.SMUB レジスタを読み出します。 print(status.measurement.current_limit.*) print(status.measurement.current_limit.condition) print(status.measurement.current_limit.event)	 B1 B2
レジスタのビットをセットします。 status.measurement.buffer_available.* = status.measurement.buffer_available.SMUA status.measurement.buffer_available.* = status.measurement.buffer_available.SMUB レジスタを読み出します。 print(status.measurement.buffer_available.*) print(status.measurement.buffer_available.condition) print(status.measurement buffer_available.event)	 B1 B2
レジスタのビットをセットします。 status.measurement.reading_overflow.* = status.measurement.reading_overflow.SMUA status.measurement.reading_overflow.* = status.measurement.reading_overflow.SMUB レジスタを読み出します。 print(status.measurement.reading_overflow.*) print(status.measurement.reading_overflow.condition) print(status.measurement.reading_overflow.event)	 B1 B2

1. * = '.ntr'、'.ptr'、および '.enable' ; smuX = smua または smub

レジスタのプログラミング例

次に示すのは、チャンネル A の電流がリミット設定値を超えたときに、装置に SRQ を発生させて TSP-Link ノードのシステムサマリビットをセットするためのコマンドシーケンス例です。

```
status.reset()                                  --全レジスタクリア
status.measurement.current_limit.enable         --電流リミットレジスタの電
= status.measurement.current_limit.SMUA           流リミットビットをイネー
                                                  ブル
status.measurement.enable =                     --ステータス測定の電流リ
status.measurement.ILMT                           ミットビットをイネーブル
status.node_enable = status.MSB                 --システムサマリをセット
                                                  （MSB をイネーブル）
status.node_enable = status.MSB                 --ステータス SRQ の MSB を
                                                  イネーブル
```

キュー

ソースメータはいずれもファーストイン - ファーストアウト（FIFO）形式である 2 つのキューを使用します。

- 出力キュー（Output Queue）— 応答メッセージを保持します。
- エラーキュー（Error Queue）— エラーメッセージとステータスメッセージを保持します。（表 B-2、p.B-3、参照）

この 2 つのキューの構成と他のレジスタとの相関関係については、ソースメータのステータスモデル（図 D-1、p.D-3）をご覧ください。

出力キュー

出力キューは装置の通常の動作に関連するデータを保持します。例えば、print コマンドが送信されると、それに対する応答メッセージが出力キューに書き込まれます。

出力キューにデータが書き込まれると、ステータスバイトレジスタセットのメッセージアベイラブル（MAV：Message Available）ビットがセットされます。メッセージが読み出されることによって出力キュー内の応答メッセージがクリアされます。キューが空になったことをもって、出力キューがリセットされたものと見なされます。出力キューが空になるとステータスバイトレジスタの MAV ビットもクリアされます。

出力キューからメッセージを読み出すためには、ソースメータをトーカとしてアドレッシングしなければなりません。

エラーキュー

エラーキューはエラーとステータスメッセージを保持します。エラーまたはステータスイベントが発生すると、そのエラー / ステータスを定義するメッセージがエラーキューに書き込まれます。

出力キューにデータが書き込まれると、ステータスバイトレジスタのエラーアベイラブル（EAV：Error Available）ビットがセットされます。そのエラー / ステータスメッセージがエラーキューから読み出されると、そのメッセージがキューからクリアされます。

エラーキューは、空になったことをもってリセットされたものと見なされます。エラーキューが空になるとステータスバイトレジスタの EAV ビットもクリアされます。

エラーキューのコントロールに使用するコマンドをまとめて表 D-11 に示します。エラーキューからメッセージを 1 個読み出すと、「一番古い」メッセージが読み出されてキューから削除されます。装置への電源投入時にエラーキューは初期化されて空になります。しかし、立ち上げ作業時に何らかの問題が検出されると、エラーキューに該当するメッセージが書き込まれます。空の状態のキューにはエラー数 0、および“No Error”メッセージが書き込まれています。

エラーキューに書き込まれるメッセージにはコード番号、メッセージテキスト、重大度、TSP-Link ノード番号が含まれています。メッセージ一覧については表 B-2（p.B-3）をご覧ください。

表 D-11
エラーキューコマンド

エラーキュー コマンド	説明
errorqueue.clear()	エラーキュー内の全エラーをクリアします。
errorqueue.count	エラー / イベントキュー内に存在するメッセージの数。
errorcode, message, severity, node = errorqueue.next()	エラーコード、テキストメッセージ、重大度、TSP-Link ノード番号を要求します。

E
速度仕様試験実施の条件

付録 E のトピックス

はじめに , E-2

使用する試験システム , E-2

概要 , E-2
- スイープ動作速度 , E-2
- シングル測定速度 , E-3
- 機能とレンジ切換え速度 , E-3
- コマンド処理 , E-4

スイープ動作速度 , E-4
- 測定値をメモリに書き込み , E-5
- 測定値を GPIB へ転送 , E-5
- ソース測定値をメモリに書き込み , E-6
- ソース測定値を GPIB へ転送 , E-6
- ソース測定合否判定をメモリに書き込み , E-6
- ソース測定合否判定を GPIB へ転送 , E-6

シングル測定速度 , E-6
- 測定値を GPIB へ転送 , E-7
- ソース測定値を GPIB へ転送 , E-7
- ソース測定合否判定を GPIB へ転送 , E-7

機能 / レンジ切換え速度 , E-8
- ソースレンジ切換え速度 , E-8
- 測定レンジ切換え速度 , E-8
- 機能切換え速度 , E-8

コマンド処理 , E-9

はじめに

この付録の目的は、付録 A として添付されている 2600 シリーズの仕様に記載されている値とほぼ同等の速度を実現するための一般的な操作方法を説明することです。ここでは、Visual Basic または VISA を呼び出して試験を実施するものとします。

使用する試験システム

PC ハードウェア：	Pentium IV 2.4 GHz、512MB RAM, National Instruments PCI-GPIB
260x 型装置：	Model 2602（試験の対象は smuA のみ）
ソフトウェア：	Microsoft Windows 2000、Microsoft Visual Basic 6.0、VISA version 3.1
ドライバ：	(PCI-GPIB) NI-488.2 version 2.2

概要

速度試験はスイープ動作速度、シングル動作速度、機能 / レンジ切換え速度、およびコマンド処理の 4 つのカテゴリーに分かれます。それぞれの試験カテゴリーごとに分けて以下の項で詳しく説明します。
スイープ動作速度試験はスクリプトを使用して、装置を多数の測定値を取得できるようにセットアップします。このスクリプトは、試験の最後に 1 つの出力（print）コマンドを実行して試験プログラムの終了を通知してくれます。シングル動作速度試験もスクリプトを使用して測定コマンドを発行します。ただし、スクリプト内部でデータを溜めるのではなく、1 回に 1 個ずつのサンプルを試験プログラムから取得します。機能 / レンジ切換え速度試験は機能またはレンジの切換えに要する時間を測定します。コマンド処理試験はコマンドを受け取ってから処理完了までに要する時間を測定します。

スイープ動作速度

スイープ動作速度試験の実行手順を以下に示します。

1. デジタル I/O コネクタのピン 1 と 2 を短絡させ、装置に電源を入れます。デジタル I/O を含む試験ではトリガをピン 2 から ピン 1 へ受け取る時間が関係するため、この両ピンを予め短絡させておきます。
2. 装置から最高の性能を引き出すため、スクリプトを実行して 260x 型を既知の試験状態に入れます。スクリプトが装置へ送られますが、この段階ではまだ実行されません。スクリプトが実行されると多数のサンプルが取り込まれます。この手法により、Visual Basic の Timer() 関数の時間分解能に起因する誤差を最小限に抑えることができます。

3. 装置を試験プログラムに同期して動作させます。これにより、前回のコマンド実行時間による影響を排除することが可能になります。装置を同期させる単純な方法は print コマンドを使用することです。

```
Private Sub Sync_Unit()
  ReceiveBuffer = ""
  TransmitBuffer = "print('done')" & CRLF
  VisaStatus = viWrite(VisaSession, TransmitBuffer, Len(TransmitBuffer), ReturnCount)
  VisaStatus = viRead(VisaSession, ReceiveBuffer, MAX_BUFFER, ReturnCount)
End Sub
```

4. タイマのスナップショットを取り込み、それからステップ 2 で説明したスクリプトを装置へ送り出して、装置が返すデータをすべて試験プログラムで取り込みます。“X to memory” 試験の場合は単一の測定結果ごとに試験プログラムへ送られ、装置側はそれによって試験が実施されたことを知ることができます。一方、“X to gpib” 速度試験ではプログラムがすべての測定データを受け取ります。以上の処理が終わるとタイマの別なスナップショットを取得します。このとき得られる開始から終了までの時間差が速度試験の結果となります。

シングル測定速度

シングル測定速度試験の操作手順は次の通りです。

1. 装置から最高の性能を引き出すため、スクリプトを実行して 260x 型を既知の試験状態に入れます。同時にこのスクリプトは 1 度に 1 個の測定値を返すように装置をセットアップします。
2. 装置を試験プログラムに同期して動作させます。これにより、前回のコマンドの実行時間とは無関係にタイミングをとることが可能になります。
3. タイマのスナップショットを取り込み、それからスクリプトを装置へ送信します。試験プログラムにはループが組み込まれており、このループが一連の測定コマンドを送り、1 度に 1 個づつの結果を取得します。ループが終了するとタイマの次のスナップショットを取得します。このとき得られる開始から終了までの時間差が速度試験の結果となります。

機能とレンジ切換え速度

機能およびレンジ切換え速度試験の操作手順は次の通りです。

1. 装置から最高の性能を引き出すため、スクリプトが 260x 型を既知の試験状態に入れます。スクリプトが装置へ送られますが、この段階ではまだ実行されません。このスクリプトは多数のサンプルを取り込むように装置を設定し、その過程で機能またはレンジを切換えます。
2. 装置を試験プログラムに同期して動作させます。これにより、前回のコマンドの実行時間とは無関係にタイミングをとることが可能になります。
3. タイマのスナップショットを取り込み、それからスクリプトを装置へ送信します。装置が測定を実行し、終了とともに試験プログラムが装置から試験完了通知を受け取ります。以上の処理が終わるとタイマの別なスナップショットを取得します。このとき得られる開始から終了までの時間差が速度試験の結果となります。

コマンド処理

コマンド処理試験の操作手順は次の通りです。

1. オシロスコープのプローブを GPIB ATN ラインに接続します。別なプローブを smua の出力へ接続します。両方の信号が適切なスケールで表示されるようにオシロスコープを設定します。
2. ここでも、スクリプトを使用して 260x 型を既知の状態に入れます。スクリプトが装置へ送られますが、この段階ではまだ実行されません。このスクリプトは装置が多数のサンプルを取り込むように装置をセットアップします。
3. スクリプトを実行して smu 出力のレベルを切換えます。
4. オシロスコープでタイミングを記録し、GPIB ATN ラインの動きが止まってから smu 出力が変化を開始するまでの時間を測定します。

スイープ動作速度

このセクションで説明する試験のうち、内部ハンドシェークが関与する試験はすべて以下の操作を含む設定用スクリプトを使用します。すべての試験は smua を使用します。

1. ソースレンジを 1V に設定し、自動レンジ切換え機能を OFF にします。
2. 測定レンジを 1V に設定し、自動レンジ切換え機能を OFF にします。
3. 測定値のフィルタ処理を OFF にします。
4. ソース機能として電圧を設定します。
5. NPLC をセットします。
6. 出力を ON にします。
7. 自動ゼロ機能を smua.AUTOZERO_ONCE に設定します。
8. バックグラウンド基準値取得のため、電圧測定値を取り込みます。
9. 自動ゼロ機能を OFF にします。
10. for- ループが希望する数のサンプルを取り込みます。
11. 測定値を取り込み、データをバッファに保存し、Lua に配列を作成してサンプルを保存します。“to GPIB” 試験の場合は、データは printnumber() 関数を使用して返されます。
12. 出力を OFF にします。
13. 試験が “to MEMORY”（メモリに書き込み）であった場合は、試験完了時に試験プログラムに対して print コマンドを発行します。

デジタル I/O ハンドシェークを使用する場合：

1. ソースレンジを 1V に設定し、自動レンジ切換え機能を OFF にします。
2. 測定レンジを 1V に設定し、自動レンジ切換え機能を OFF にします。
3. 測定値のフィルタ処理を OFF にします。

4. ライン 1 のトリガモードを FALLING に設定します。digio.trigger[1].mode = digio.TRIG_FALLING
5. ライン 2 のパルス幅を設定します。digio.trigger[2].pulsewidth = 1E-6
6. ソース機能として電圧を選択します。
7. NPLC をセットします。
8. 出力を ON にします。
9. 自動ゼロ機能を smua.AUTOZERO_ONCE に設定します。
10. バックグラウンド基準値取得のための内部処理として測定を行います。
11. 自動ゼロ機能を OFF にします。
12. for- ループを使用して希望する数のサンプルを取り込みます。
13. ループが一連の測定値を生成します（各測定の前に digio.trigger[1].wait を発行）。電圧測定値をバッファに保存します。“to GPIB”試験の場合は、データは printnumber() 関数を使用して返されます。
最後に digio.trigger[2].wait を送ります。
14. 出力を OFF にします。
15. 試験が “to MEMORY”（メモリに書き込み）であった場合は、試験完了時に試験プログラムに対して print コマンドを発行します。

測定値をメモリに書き込み

スクリプトは smua.measure.v(nvbuffer1) を使用して測定値を取り込みます。for- ループを使用してデータを取り込む代わりに、測定値のカウント数として非常に大きな値（たとえば 1,000）を設定します。測定結果は Lua 内に作成した配列ではなく、nvbuffer1 に書き込まれます。

Visual Basic を使用して速度計測を行うプログラム例を次に示します。TestScript() はセットアップ用のスクリプトです。

```
tstart = Timer‘ Starts timer
   TransmitBuffer = "TestScript()" & CRLF
   ReceiveBuffer = ""
   VisaStatus = viWrite(VisaSession, TransmitBuffer, Len(TransmitBuffer), ReturnCount)
   VisaStatus = viRead(VisaSession, ReceiveBuffer, MAX_BUFFER, ReturnCount)
tstop = Timer  ‘Stops timer

Result = NumPoints/ (tstop – start)  ‘ NumPoints is the number of samples taken
```

測定値を GPIB へ転送

この試験も「測定値をメモリに書き込み」によく似ていますが、データをバッファに書き込む代わりに毎回の測定ごとに結果を試験プログラムに返します。また、返されるデータの形式を ASCII ではなく SREAL に設定します。実際にデータを返すのは printnumber() コマンドです。

Visual Basic を使用して測定結果を取得するプログラム例を次に示します。TestScript() はセットアップ用のスクリプトです。

```
tstart = Timer
   TransmitBuffer = "TestScript()" & CRLF
   VisaStatus = viWrite(VisaSession, TransmitBuffer, Len(TransmitBuffer), ReturnCount)
   For i = 1 To NumPoints‘ Number of samples to take
      VisaStatus = viRead(VisaSession, ReceiveBuffer, MAX_BUFFER, ReturnCount)
   Next i
tstop = Timer

Result = NumPoints/ (tstop – start)  ‘ NumPoints is the number of samples taken
```

ソース測定値をメモリに書き込み

この試験で使用するセットアップは「測定値をメモリに書き込み」の場合とよく似ています。

ただし、ここでは smua.measure.v() の代わりに smua.measurevandstep(Levelv) 関数を使用します（セクション 12 のコマンド解説参照）。Levelv の値を段階的に増大させながら試験を行い、データを Lua 内に作成した配列に保存します。

ソース測定値を GPIB へ転送

この試験のセットアップは「測定値を GPIB へ転送」によく似ていますが、相違点は smua.measure.v() の代わりに smua.measurevandstep(Levelv) 関数を使用することです。Levelv を段階的に増大させながら試験を行います。

ソース測定合否判定をメモリに書き込み

この試験は「ソース測定をメモリに書き込み」試験によく似ていますが、いくつかの論理処理が追加されています。測定値をある値と比較し、測定値がしきい値を超えたか否かを判定してフラグを 1 または 0 にセットします。

ソース測定合否判定を GPIB へ転送

この試験のセットアップは「ソース測定値を GPIB へ転送」とよく似ていますが、いくつかの論理処理が追加されています。測定値をある値と比較し、測定値がしきい値を超えたか否かを判定してフラグを 1 または 0 にセットします。

シングル測定速度

このセクションで説明したそれぞれの試験は以下の項目を実行するセットアップ用スクリプトを使用しています。

1. ソースレンジを 1V に設定し、自動レンジ切換え機能を OFF にします。
2. 測定レンジを 1V に設定し、自動レンジ切換え機能を OFF にします。
3. 測定値のフィルタ処理を OFF にします。
4. ソース機能として電圧を選択します。
5. NPLC をセットします。
6. 出力を ON にします。
7. 自動ゼロ機能を smua.AUTOZERO_ONCE に設定します。

8. バックグラウンド基準値取得のための内部処理として測定値を取り込みます。
9. 自動ゼロ機能を OFF にします。
10. format.data コマンドを SREAL 形式用にセットします。
11. 関数を実行して測定値を 1 つ取り込みます。
12. printnumber() 関数を使用してデータを返します。
13. 出力を OFF にします。

測定値を GPIB へ転送

smua.measure.v() 関数を使用して 1 個の電圧測定値を記録します。

Visual Basic を使用して測定結果を取得するプログラム例を次に示します。TestScript() はセットアップ用のスクリプトです。

```
TransmitBuffer = "printnumber(smua.measure.v())" & CRLF
tlen = Len(TransmitBuffer)

tstart = Timer
    For i = 1 To NumPoints ‘ NumPoints is the number of samples taken
        VisaStatus = viWrite(VisaSession, TransmitBuffer, tlen, ReturnCount)
        VisaStatus = viRead(VisaSession, ReceiveBuffer, MAX_BUFFER, ReturnCount)
     Next i
tstop = Timer

Result = NumPoints/ (tstop – start)
```

ソース測定値を GPIB へ転送

この試験は「測定値を GPIB へ転送」によく似ていますが、相違点は smua.measure.v() の代わりに smua.measurevandstep(Levelv) 関数を使用することです。Levelv の値を段階的に増大させながら試験を行います。

ソース測定合否判定を GPIB へ転送

この試験のセットアップは「ソース測定値を GPIB へ転送」とよく似ていますが、いくつかの論理処理が追加されています。すなわち、毎回の測定値をある値と比較し、測定値がしきい値を超えたか否かを判定してフラグを 1 または 0 にセットします。

機能 / レンジ切換え速度

このセクションで説明する試験は以下のセットアップを使用します。

1. ソースレンジを 1V に設定し、自動レンジ切換え機能を OFF にします。
2. 測定レンジを 1V に設定し、自動レンジ切換え機能を OFF にします。
3. 測定値のフィルタ処理を OFF にします。
4. ソース機能として電圧を選択します。
5. NPLC をセットします。
6. 出力を ON にします。
7. 機能のレンジを変更します。
8. 出力を OFF にします。

ソースレンジ切換え速度

この試験は前項で説明したセットアップを使用して、2 種類のソースレンジ間で切換えを行います。

Visual Basic を使用して測定結果を取得するプログラム例を次に示します。TestScript() はセットアップ用のスクリプトです。

```
tstart = Timer
   TransmitBuffer = "TestScript()" & CRLF
   ReceiveBuffer = ""
   VisaStatus = viWrite(VisaSession, TransmitBuffer, Len(TransmitBuffer), ReturnCount)
   VisaStatus = viRead(VisaSession, ReceiveBuffer, MAX_ BUFFER, ReturnCount)
 tstop = Timer

Result = NumPoints/ (tstop – start)  ‘ NumPoints is the number of samples taken
```

測定レンジ切換え速度

この試験メソッドは「ソースレンジ切換え速度」と基本的に同じものですが、試験の途中で 2 つの測定レンジ間の切換えが行われる点が異なっています。

機能切換え速度

この試験メソッドは「ソースレンジ切換え速度」と基本的に同じものですが、試験の途中で電圧と電流機能の切換えが行われる点が異なっています。

コマンド処理

この試験では以下のセットアップを使用します。

1. ソースレンジを 1V に設定し、自動レンジ切換え機能を OFF にします。
2. ソース機能として電圧を選択します。
3. 出力を ON にします。
4. 一連の smua.source.levelv() コマンドを送ることにより smu の電圧レベルを 1V と 0V の間で変化させてゆきます。
5. オシロスコープでタイミングを記録し、GPIB ATN ラインの動きが止まってから smu 出力が変化を開始するまでの時間を測定します。
6. 出力を OFF にします。

KEITHLEY

サービス請求フォーム

モデル型式 _____シリアル番号 _____日付_____

お名前、電話番号、FAX _____

企業名 _____

コントロール設定および発生した問題点を説明してください。問題点に該当する項目にチェックマークを付けてください。_____

_____

_____

❑ 時折問題が発生する ❑ アナログ出力がディスプレイに追随する
❑ 特定のレンジ / 機能で問題が発生する、具体的に：

_____

❑ IEEE が機能しない ❑ 電源立ち上げ時に問題発生 ❑ バッテリとヒューズは問題なし
❑ 前面パネルは機能している ❑ すべてのレンジ / 機能で動作不良 ❑ ケーブルはすべてチェック済み

ディスプレイ / 出力（1 つだけ選択してください）

❑ ドリフト ❑ ゼロ調節不能 ❑ 不安定
❑ 過負荷 ❑ 印加した入力を読み込めない

修理後の校正について：

❑ 校正のみ ❑ 校正証明書要 ❑ 校正データ要

（追加項目があれば別紙に記入して添付してください）

接続されているすべての装置を含めた測定システムのブロック図を示してください（通電の有無も記入してください）。信号源についても明記してください。

測定実施場所（工場、環境をコントロールしたラボ、屋外等々）_____

_____

使用電源電圧 _____ 周囲温度 _____℃

相対湿度 _____%RH その他 _____

その他の情報（ユーザが独自の改造を施している場合は具体的に説明してください）。

_____

_____

お名前と電話番号が間違いなく記入されていることを確認してください。

本書の内容は予告なく変更されることがあります。
Keithley の商標はすべて Keithley Instruments, Inc に帰属します。
その他の商標は各所有会社に帰属します。

KEITHLEY

Belgium: Sint-Pieters-Leeuw・02-363 00 40・Fax: 02-363 00 64・www.keithley.nl
China: Beijing・8610-5501-0010・Fax: 8610-8225-5018・www.keithley.com.cn
Finland: Helsinki・09-5306-6560・Fax: 09-5306-6565・www.keithley.com
France: Saint-Aubin・01-64 53 20 20・Fax: 01-60 11 77 26・www.keithley.fr
Germany: Germering・49-89-84-93-070・Fax: 49-89-84-93-0734・www.keithley.de
Great Britain: Theale・44-118 -929 7500・Fax: 44-118-929-7519・www.keithley.co.uk
India: Bangalore・91-80-2212-8027・Fax : 91-80-2212-8005・www.keithley.com
Italy: Milano・39-02-55 38 42・Fax: 39-02-55 38 42 28・www.keithley.it
日本: 東京・03-5733-7555・Fax: 03-5733-7556・www.keithley.jp
Korea: Seoul・82-2-574-7778・Fax: 82-2-574-7838・www.keithley.com
Netherlands: Gorinchem・0183-635333・Fax: 0183-630821・www.keithley.nl
Singapore: Singapore・65-6747-9077・Fax: 65-6747-2991・www.keithley.com
Sweden: Solna・08-509 04 600・Fax: 08-655 26 10・www.keithley.com
Taiwan: Hsinchu・886-3-572-9077・Fax: 886-3-572-9031・www.keithley.com.tw